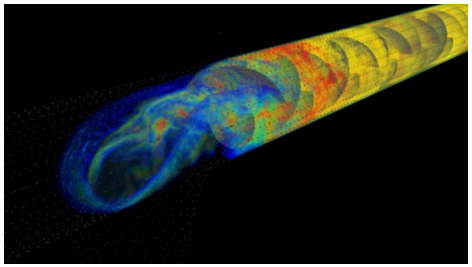
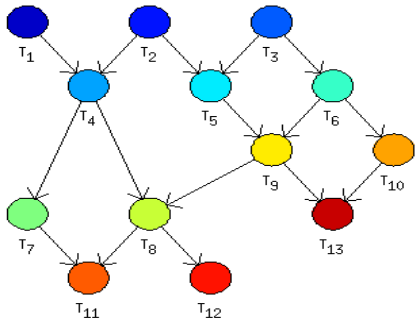
Software Abstractions for Extreme-Scale Scalability of Computational Frameworks



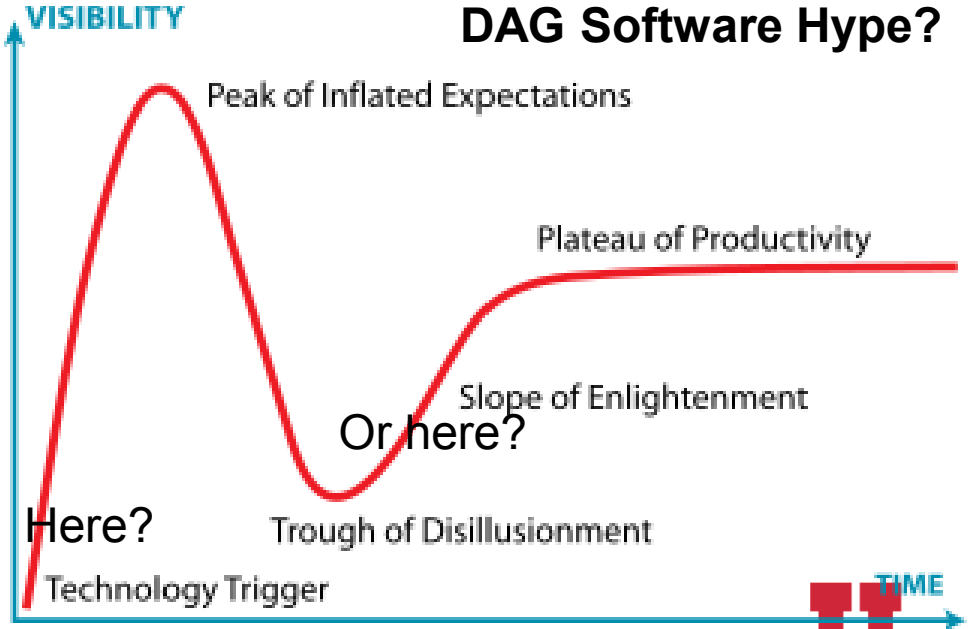
Martin Berzins

www.uintah.utah.edu

1. Background, motivation, Directed Acyclic Graph software
2. A DAG Example the Uintah Software
3. Engineering for Scalability with DAGs
4. Conclusions



DAG Software Hype?



DOE ASCI (97-10), NSF , DOE NETL+NNSA ARL
NSF , INCITE, XSEDE

SCI INSTITUTE

THE UNIVERSITY OF UTAH

BACKGROUND
MOTIVATION DAGs, SOFTWARE

Extreme Scale Research and teams in Utah

Energetic Materials: Chuck Wight, Jacqueline Beckvermit, Joseph Peterson, Todd Harman, Qingyu Meng NSF PetaApps 2009-2014 $1M, P.I. MB

PSAAP Clean Coal Boilers: Phil Smith (P.I.), Jeremy Thornock James Sutherland etc Alan Humphrey John Schmidt DOE NNSA 2013-2018 $16M (MB Cs lead)

Electronic Materials by Design: MB (PI) Dmitry Bedrov, Mike Kirby, Justin Hooper, Alan Humphrey Chris Gritton, + ARL TEAM 2011-2016 $12M

Software team:

Qingyu Meng*, John Schmidt, Alan Humphrey, Justin Luitjens**,

DSL team lead

James Sutherland





* Now at Google

** Now at NVIDIA

The Exascale challenge for Future Software?

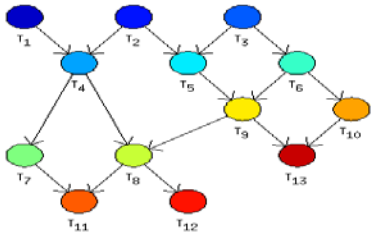
Harrod SC12: "today's bulk synchronous (BSP), distributed memory, execution model is approaching an efficiency, scalability, and power wall."

Compute

Communicate

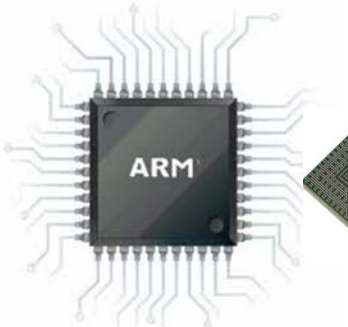
Compute

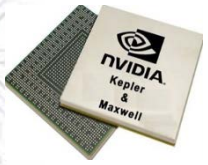
Sarkar et al. "Exascale programming will require prioritization of critical-path and non-critical path tasks, adaptive directed acyclic graph scheduling of critical-path tasks, and adaptive rebalancing of all tasks ..."
" DAG Task-based programming has always been a bad idea. It was a bad idea when it was introduced and it is a bad idea now " Parallel Processing Award Winner



Much architectural uncertainty, many storage and power issues. Adaptive portable software needed

Power needs force use of accelerators





Some Historical Background

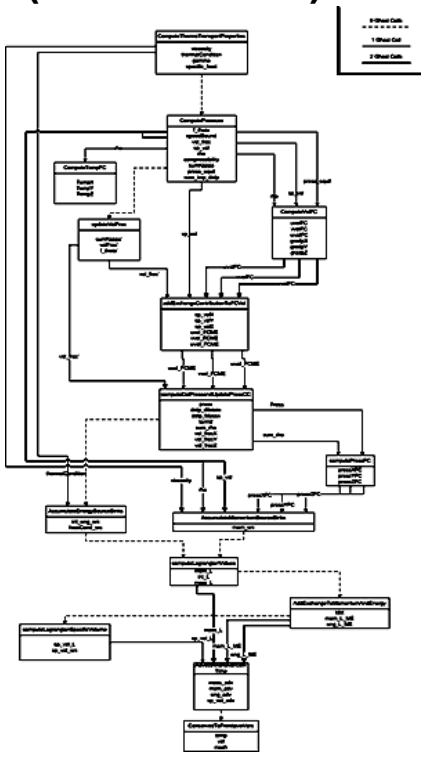
- Vivek Sarkar's thesis (1989)
  - Graphical rep. for parallel programs
  - Cost model
  - Compile time cost assignment
  - Macro-data flow for execution
  - Compile time schedule
  - Prototype implementation 20 processors
- Charm++ Sanjay Kale et al. 1990s onward
- Uintah Steve Parker 1998 onward

Present Day

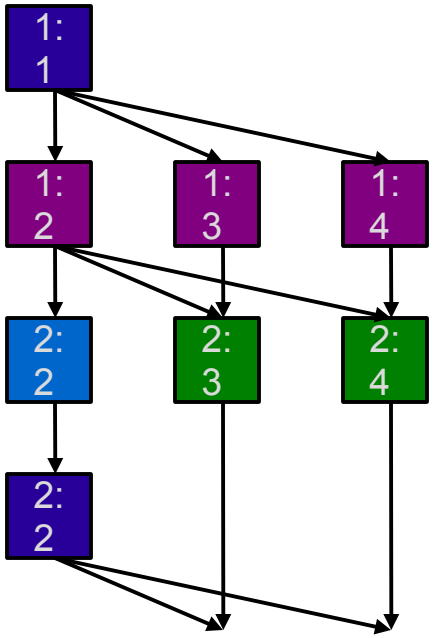
Much work on task graphs –
e.g. O. Sinnen "Task Scheduling for Parallel Systems"

Task Graph Based Languages/Frameworks

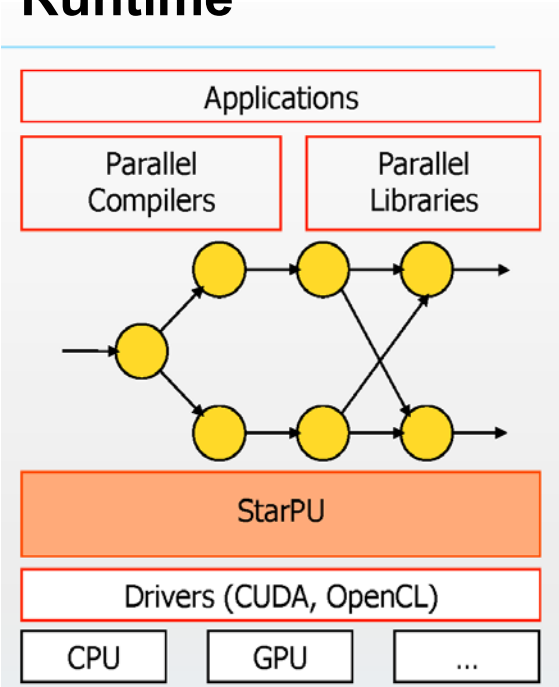
Uintah Taskgraph based PDE Solver (Parker 1998)

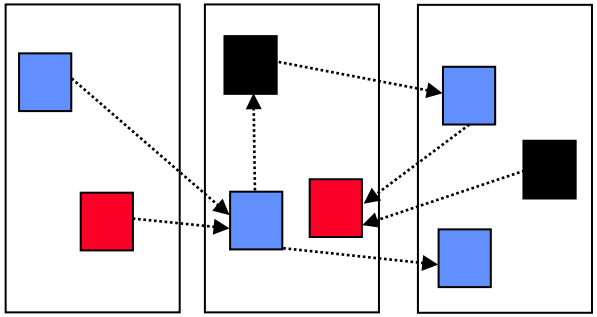


Plasma (Dongarra): DAG based Parallel linear algebra software



StarPU Task Graph Runtime

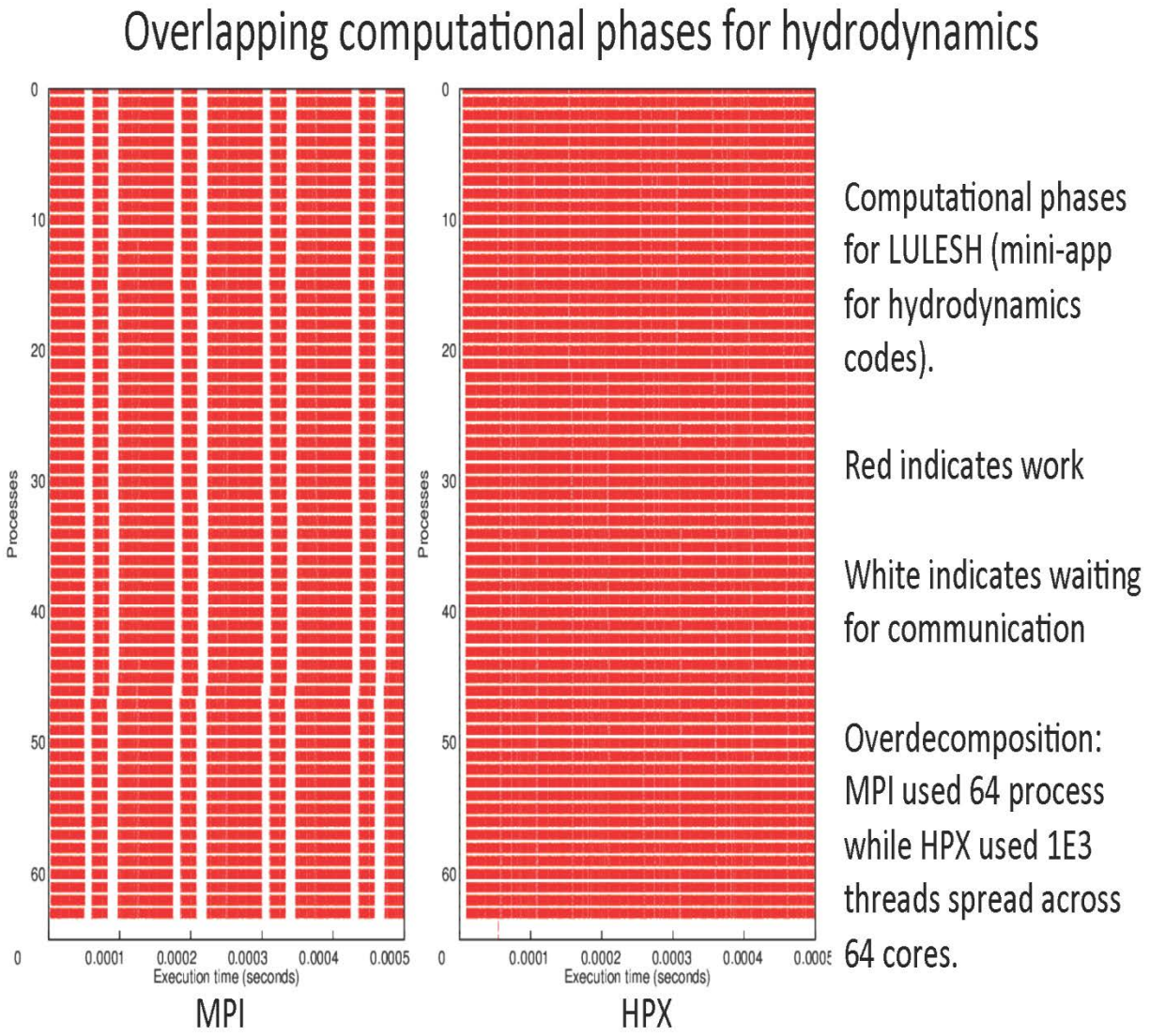




Kale (1990) Charm++: Object-based Virtualization

Why does Dynamic Execution of Directed Acyclic Graphs Work Well?

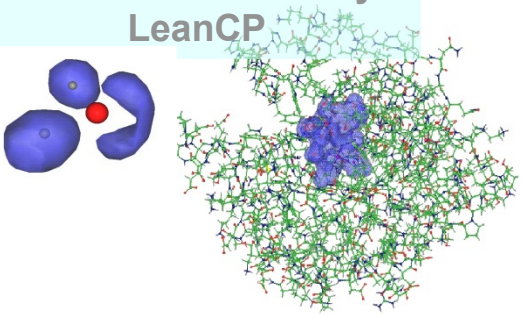
- Eliminate spurious synchronizations points
- Have multiple task-graphs per multicore (+ gpu) node – provides excess parallelism - slackness
- Overlap communication with computation by executing tasks as they become available – avoid waiting (use out-of order execution).
- Load balance complex workloads by having a sufficiently rich mix of tasks per multicore node that load balancing is done per node



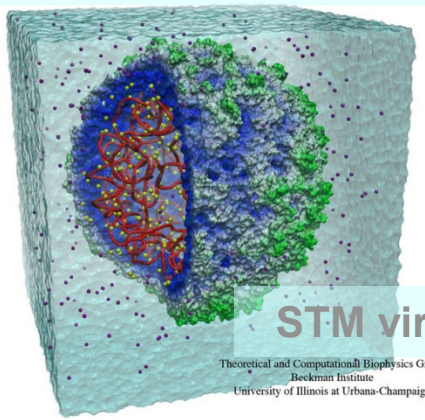
Sterling et al. Express Project - faster?

Develop abstractions in context of full-scale applications

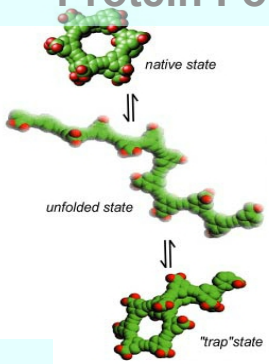
Quantum Chemistry
LeanCP



NAMD: Molecular Dynamics

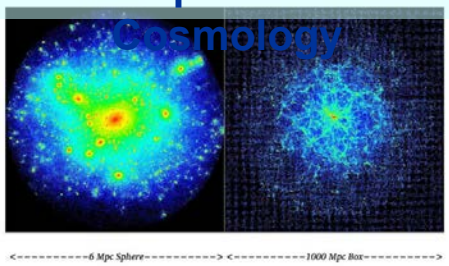


Protein Folding



STM virus simulation

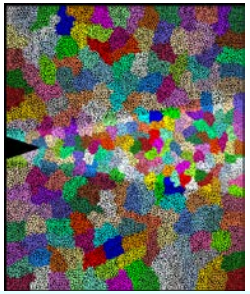
Computational
Cosmology



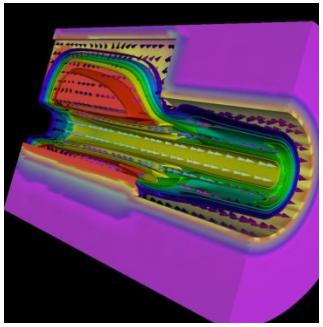
Parallel Objects,

Adaptive Runtime System

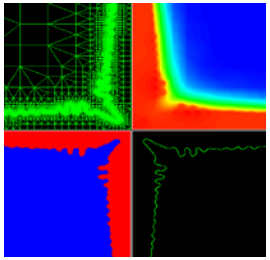
Libraries and Tools



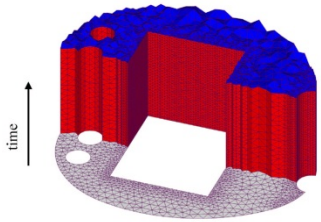
Crack Propagation



Rocket Simulation



Dendritic Growth

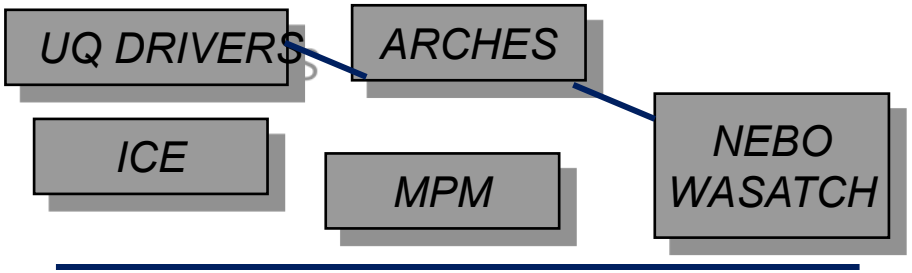


Space-time meshes

APPLICATIONS CHARM++ [SOURCE: KALE]

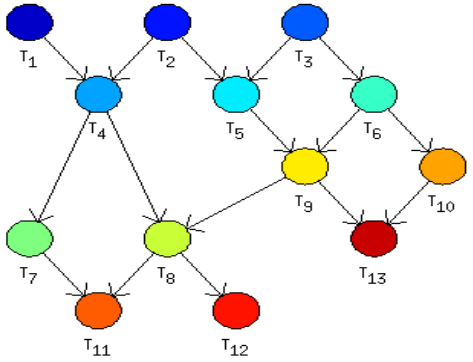
UINTAH FRAMEWORK

Some components have not changed as we have gone from 600 to 600K cores

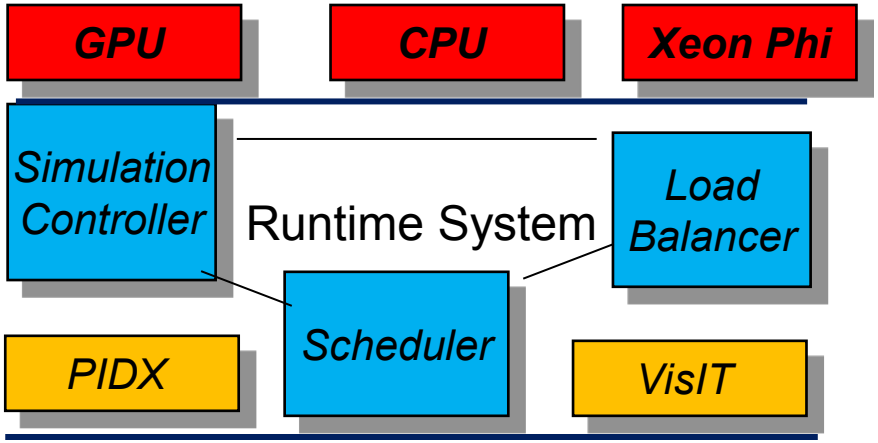


- **Application Specification** via ICE MPM ARCHES or NEBO/WASATCH DSL

- **Abstract task-graph** program that



- Is compiled for

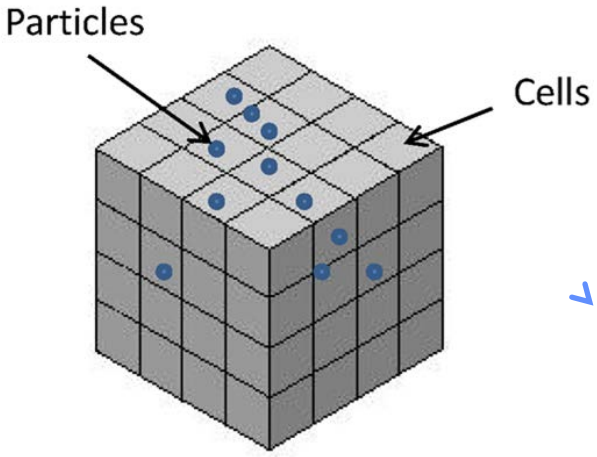


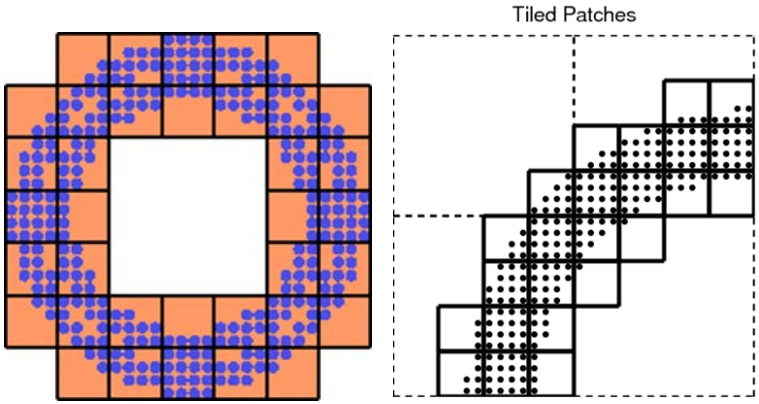
- Executes on: **Runtime System** with: asynchronous out-of-order execution, work stealing, Overlap communication & computation.Tasks running on cores and accelerators
- **Scalable I/O via Visus PIDX**

Uintah(X) Architecture Decomposition

Uintah Patch, Variables and AMR Outline

ICE is a cell-centered finite volume method for Navier Stokes equations





- **Structured Grid + Unstructured Points**
- **Patch-based Domain Decomposition**
- **Regular Local Adaptive Mesh Refinement**

ICE Structured Grid Variable (for Flows) are Cell Centered Nodes, Face Centered Nodes.

Unstructured Points (for Solids) are **MPM** Particles

- **Dynamic Load Balancing**
  - **Profiling + Forecasting Model**
  - **Parallel Space Filling Curves**
- **Works on MPI and/or thread level**

ARCHES is a combustion code using several different radiation models and linear solvers

Uintah:MD based on Lucretius is a new molecular dynamics component

Burgers Example I

```
<Grid>
  <Level>
    <Box label = "1">
    <lower>      [0,0,0]       </lower>
    <upper>      [1.0,1.0,1.0] </upper>
    <resolution> [50,50,50]    </resolution>
    <patches>    [2,2,2]       </patches>
    <extraCells> [1,1,1]       </extraCells>
   </Box>
  </Level>
</Grid>
```

25 cubed patches
8 patches
One level of halos

```
void  Burger::scheduleTimeAdvance( const LevelP& level,
                                   SchedulerP& sched)
{
    ..
  task->requires(Task::OldDW, u_label, Ghost::AroundNodes, 1);
  task->requires(Task::OldDW, sharedState_->get_delt_label());

  task->computes(u_label);
  sched->addTask(task, level->eachPatch(), sharedState_->allMaterials());
}
```

Get old solution from old data warehouse
One level of halos
Compute new solution

Burgers Equation code

$$U_t + UU_x = 0$$

```
void Burger::timeAdvance(const ProcessorGroup*,    const PatchSubset* patches,
 const MaterialSubset* matls,  DataWarehouse* old_dw, DataWarehouse* new_dw)
 //Loop for all patches on this processor
 { for(int p=0;p<patches->size();p++){
//Get data from  data warehouse including 1 layer of  "ghost" nodes from
    surrounding patches
    old_dw->get(u, lb_->u, matl, patch, Ghost::AroundNodes, 1);

   // dt, dx Time and space increments
   Vector dx = patch->getLevel()->dCell();
      old_dw->get(dt, sharedState_->get_delt_label());

  // allocate memory for results new_u
      new_dw->allocateAndPut(new_u, lb_->u, matl, patch);

 // define iterator range  l and h        lots missing here  and Iterate through all the
    nodes
   for(NodeIterator iter(l, h);!iter.done(); iter++){
     IntVector n = *iter;
     double dudx = (u[n+IntVector(1,0,0)] - u[n-IntVector(1,0,0)]) /(2.0 * dx.x());
      double du = - u[n] * dt * (dudx);
     new_u[n]= u[n] + du;
   }
```

Uintah Directed Acyclic (Task) Graph-Based Computational Framework

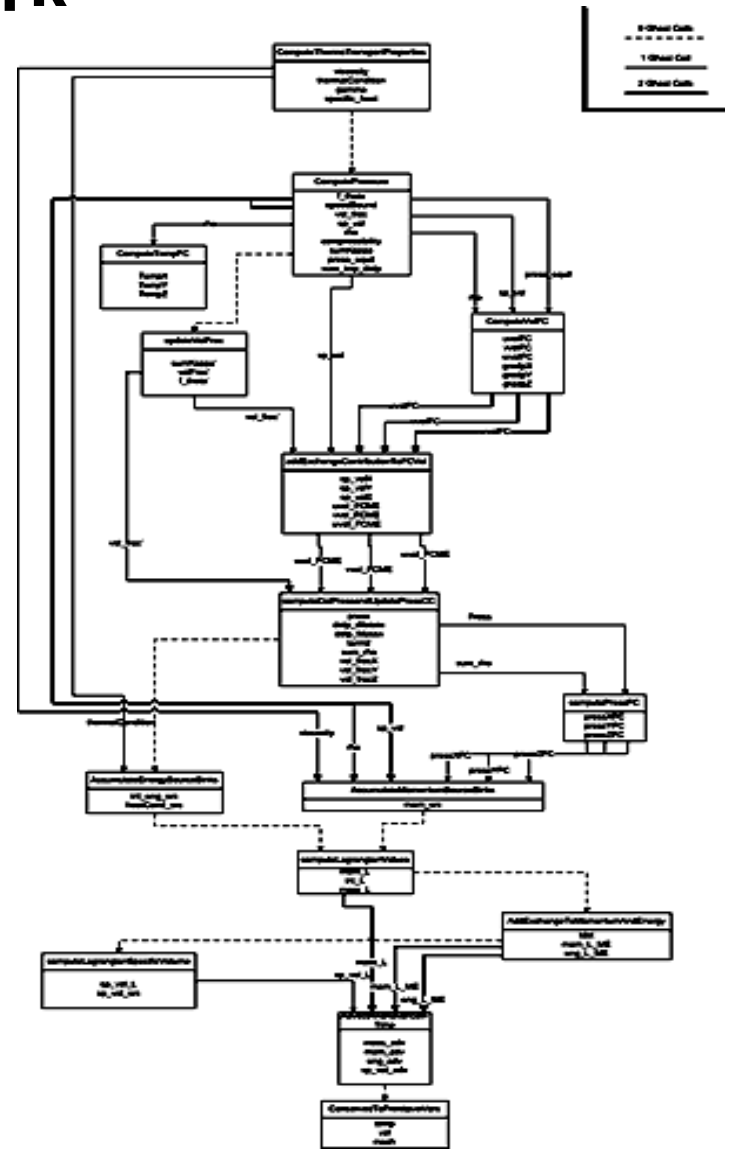
Each task defines its computation with required inputs and outputs

Uintah uses this information to create a task graph of computation (nodes) + communication (along edges)

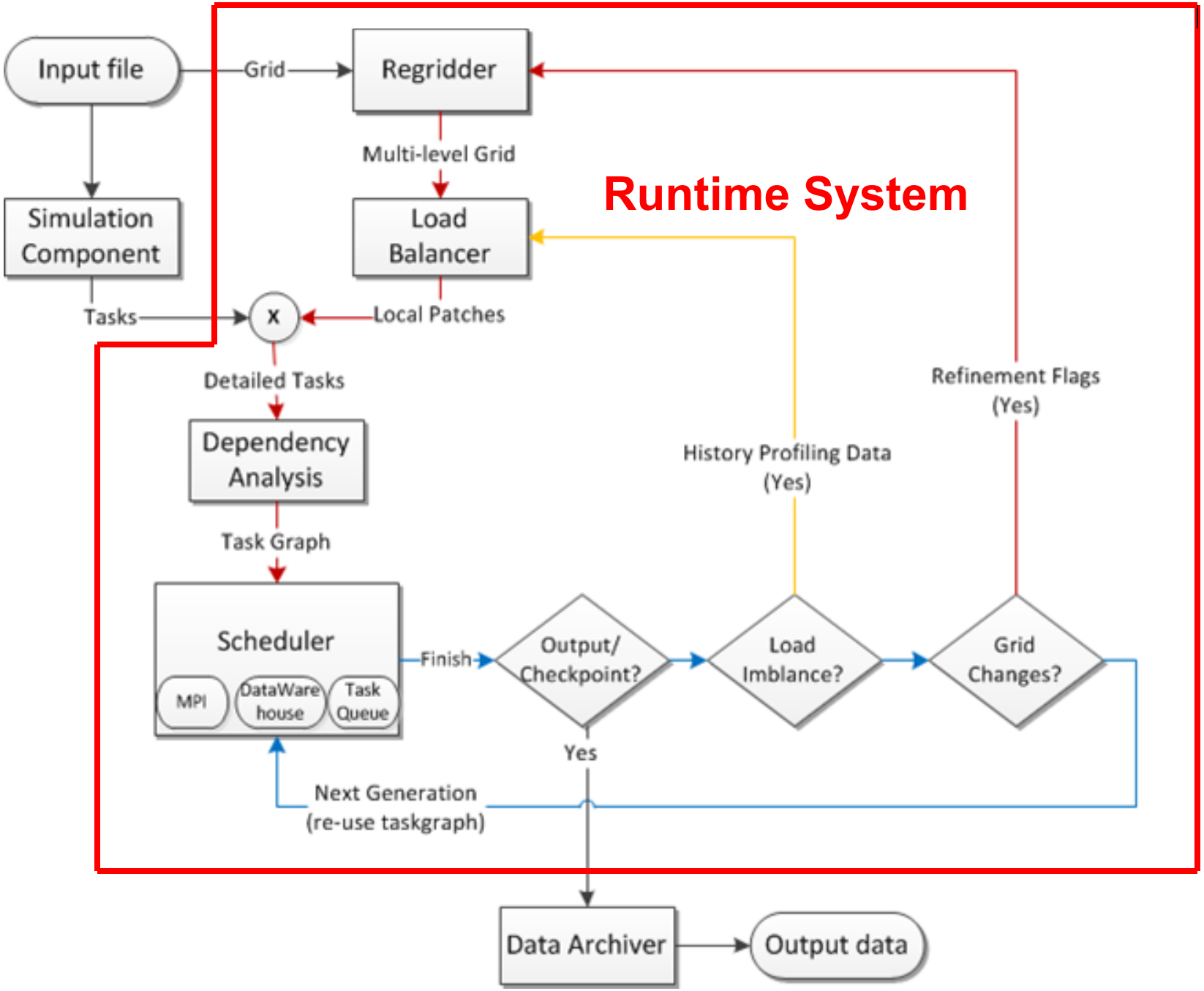
Tasks do not explicitly define communications but only what inputs they need from a data warehouse and which tasks need to execute before each other.

Communication is overlapped with computation

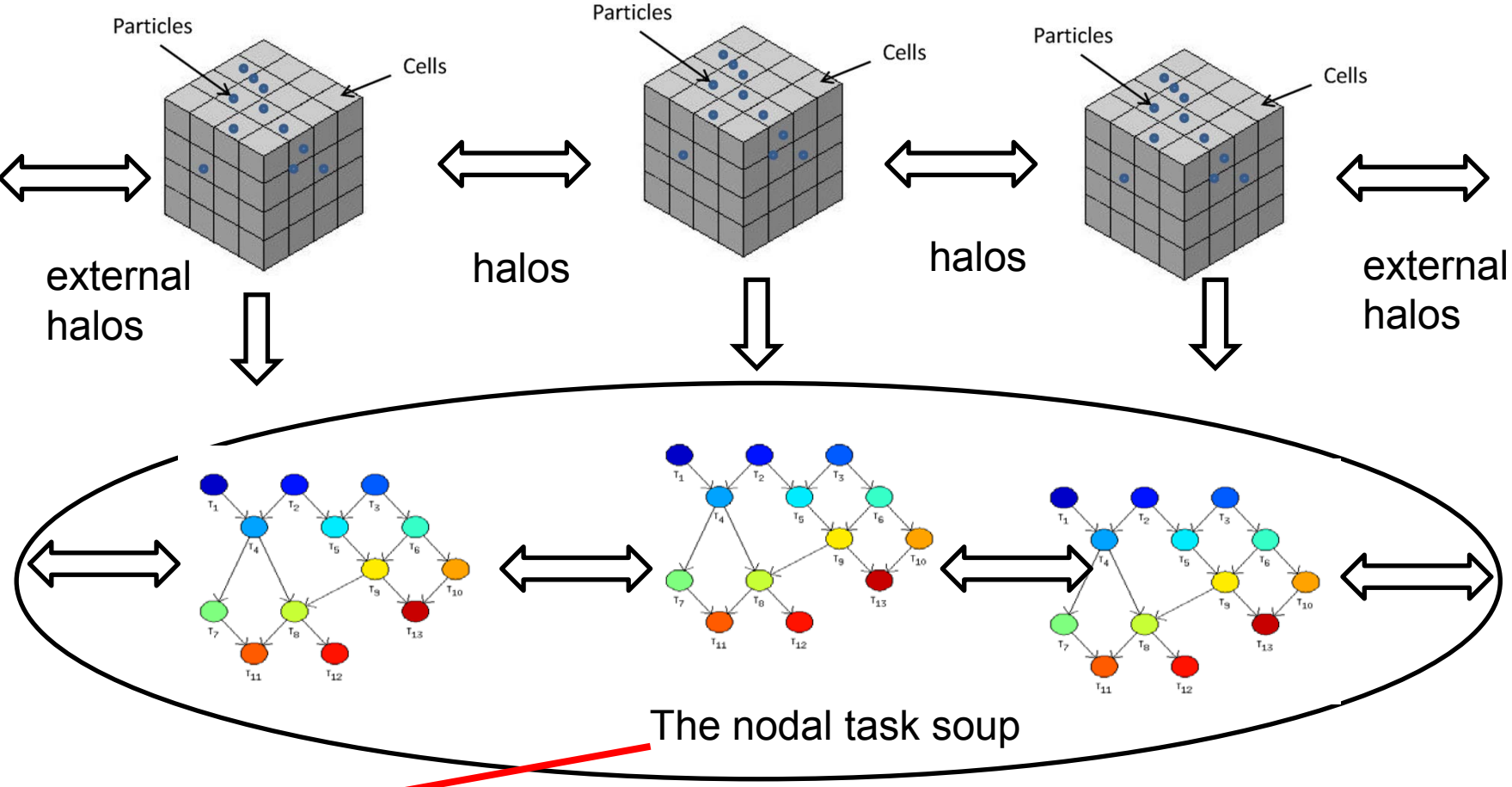
Taskgraph is executed adaptively and sometimes out of order, inputs to tasks are saved

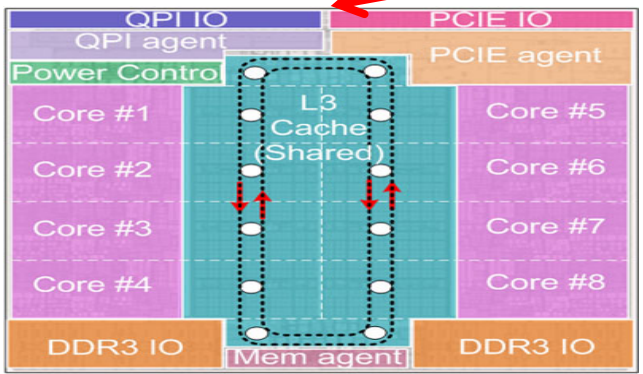


Tasks get data from OLD Data Warehouse and put results into NEW Data Warehouse


Input file
Grid
Regridder
Multi-level Grid
Runtime System
Simulation Component
Load Balancer
Tasks
X
Local Patches
Detailed Tasks
Refinement Flags (Yes)
Dependency Analysis
History Profiling Data (Yes)
Task Graph
Scheduler
MPI
DataWare house
Task Queue
Finish
Output/ Checkpoint?
Load Imblance?
Grid Changes?
Yes
Next Generation (re-use taskgraph)
Data Archiver
Output data

Task Graph Structure on a Multicore Node with multiple patches





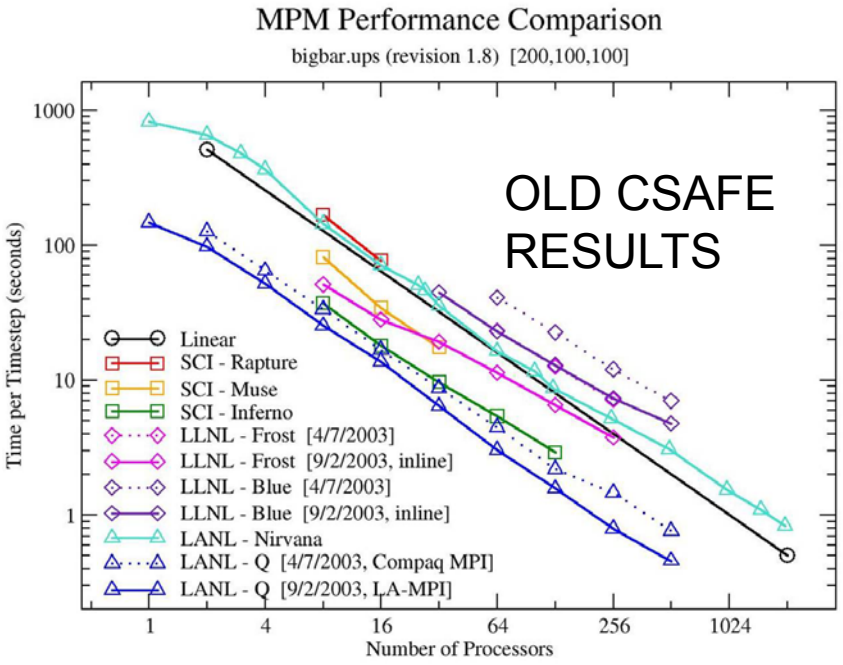
This is not a single graph. Multiscale and Multi-Physics merely add flavor to the "soup".

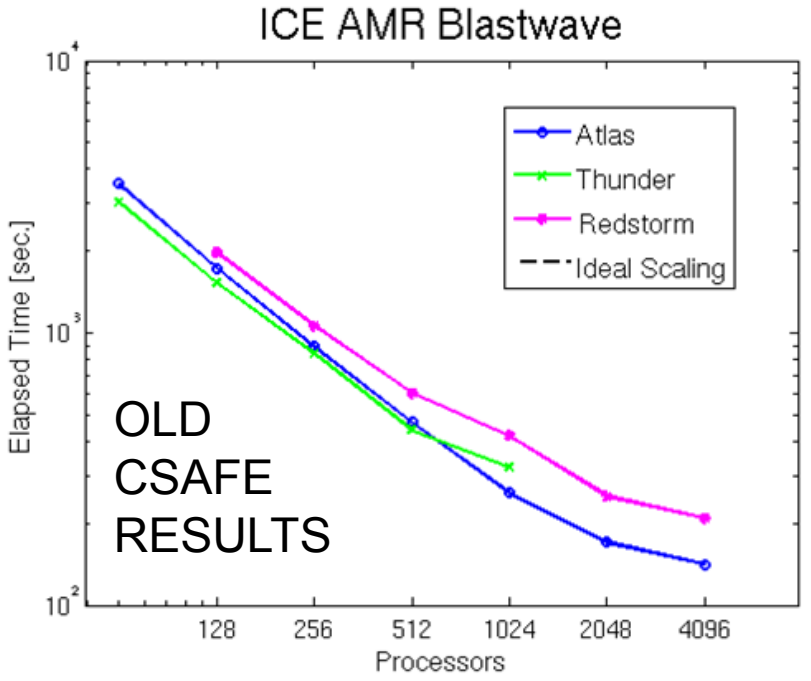
The DAG Approach is not a silver bullet

Uintah Phase 1 1998-2005 – overlap communications with computation. Static task graph execution standard data structures one MPI process per core. No AMR.

Uintah Phase 2 2005-2010 improved fast data structures, load balancer. AMR to 12k cores, then 100K cores using new approach before data structures cause problems. Out of order and dynamic task execution.

Uintah Phase 3 2010- Hybrid model. Theaded runtine system on node. One MPI process and one data warehouse per node. Multiple cores and GPUs grab tasks as needed. Fast scalable use of hypre for linear equations.

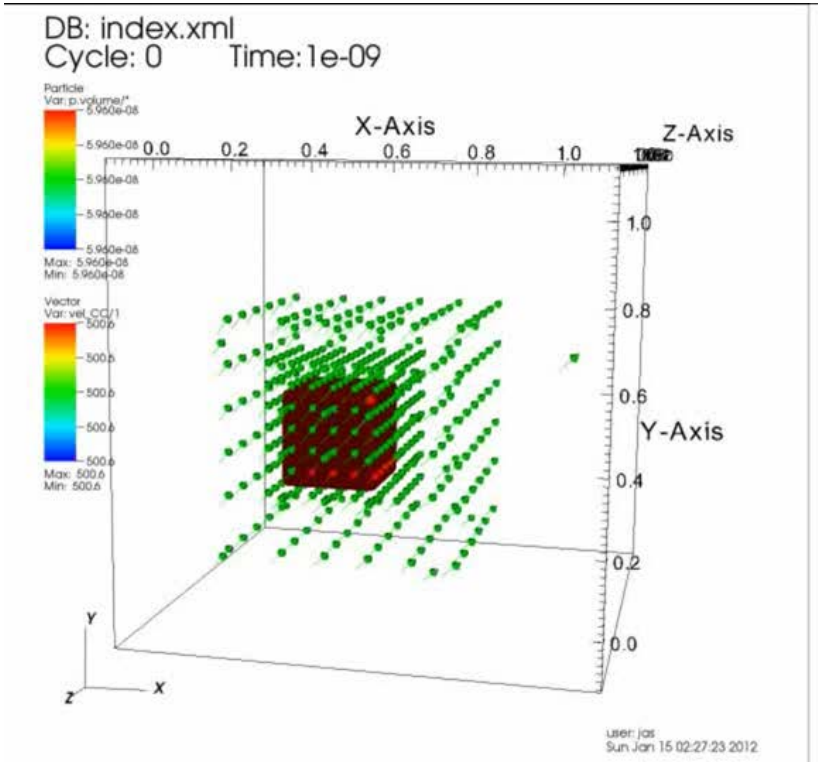




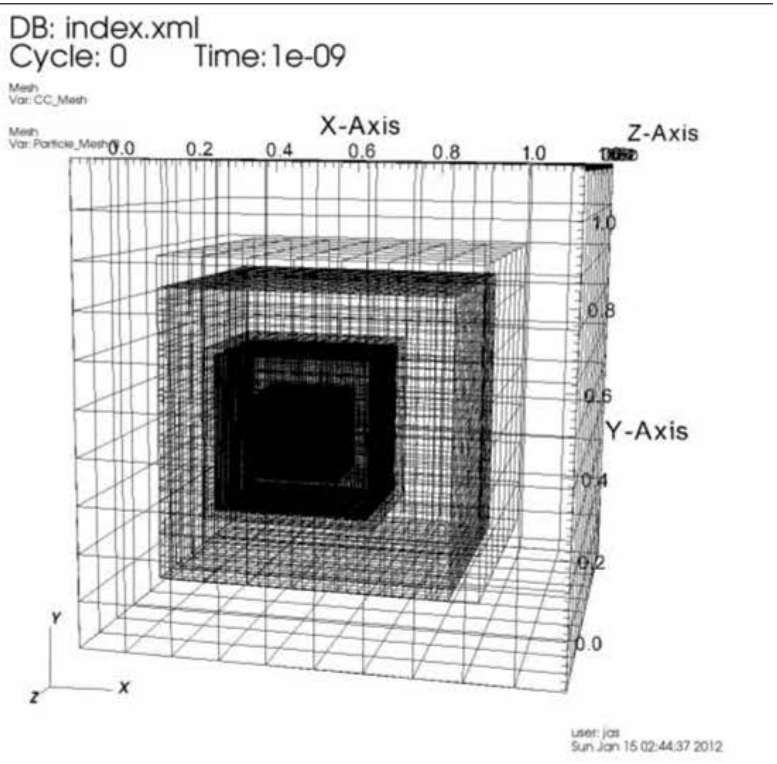
UINTAH SCALABILITY

Explosives Problem 1 Fluid-Structure Benchmark Example: AMR MPMICE

A PBX explosive flow quickly pushing a piece of its metal container



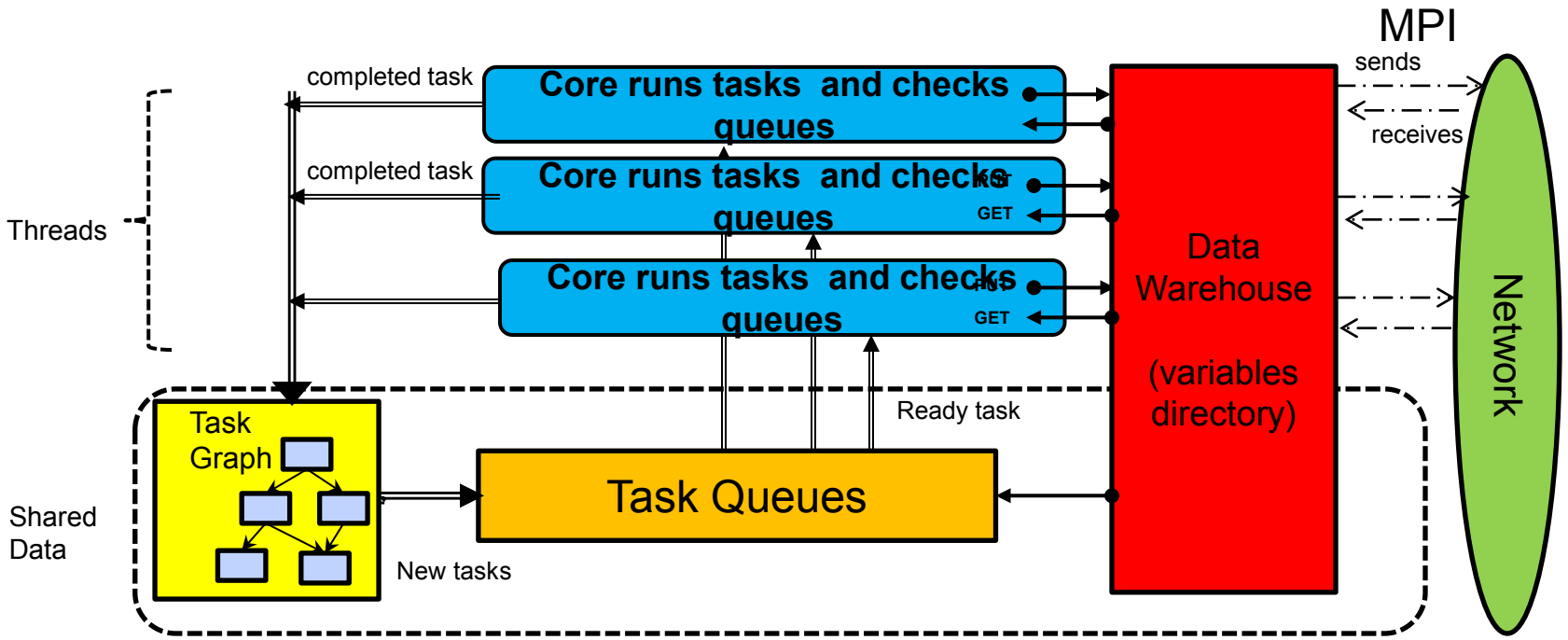
Flow velocity and particle volume



Computational grids and particles

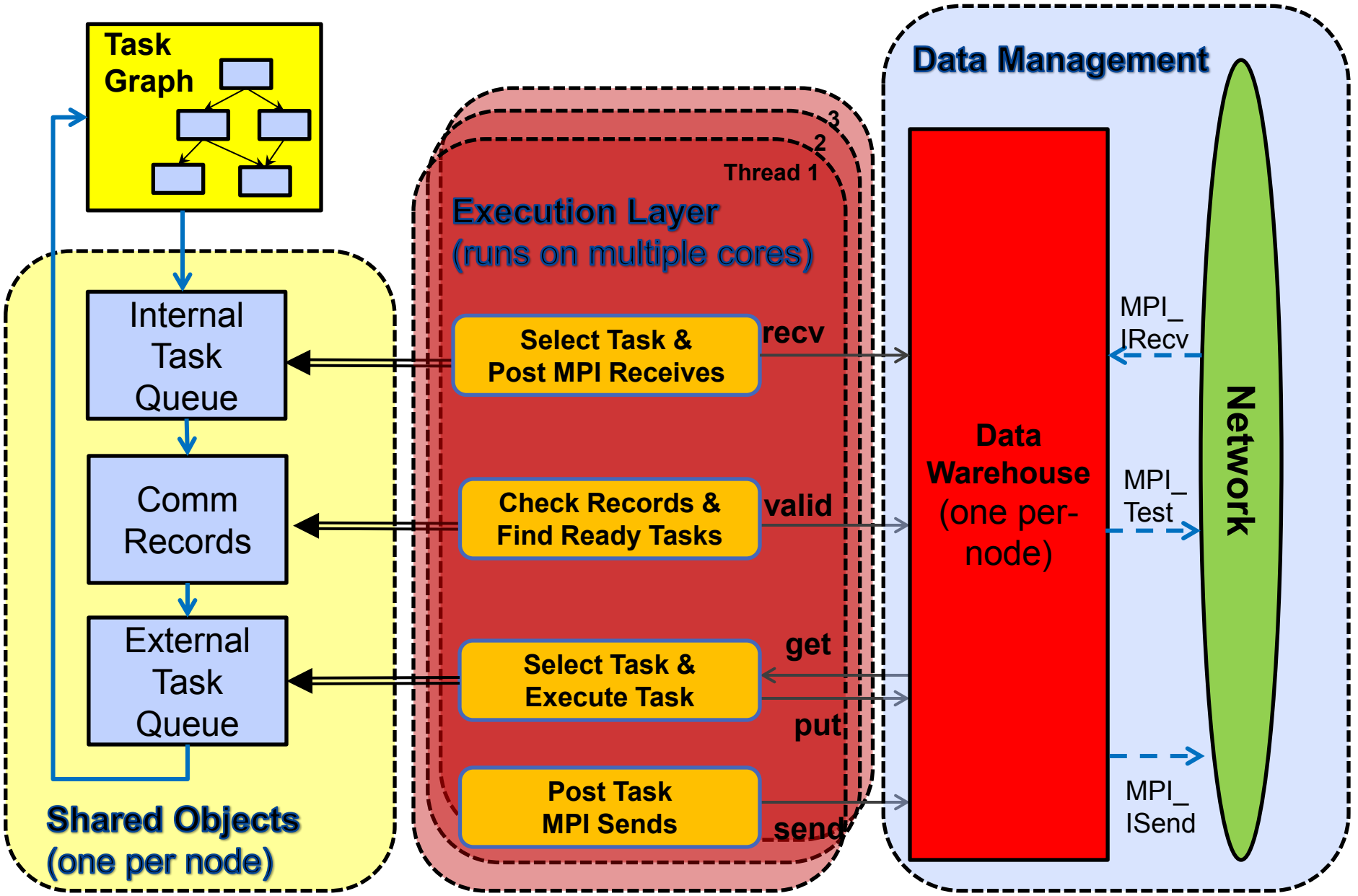
Grid Variables: Fixed number per patch, relative easy to balance
Particle Variables: Variable number per patch, hard to load balance

Thread/MPI Scheduler (De-centralized)

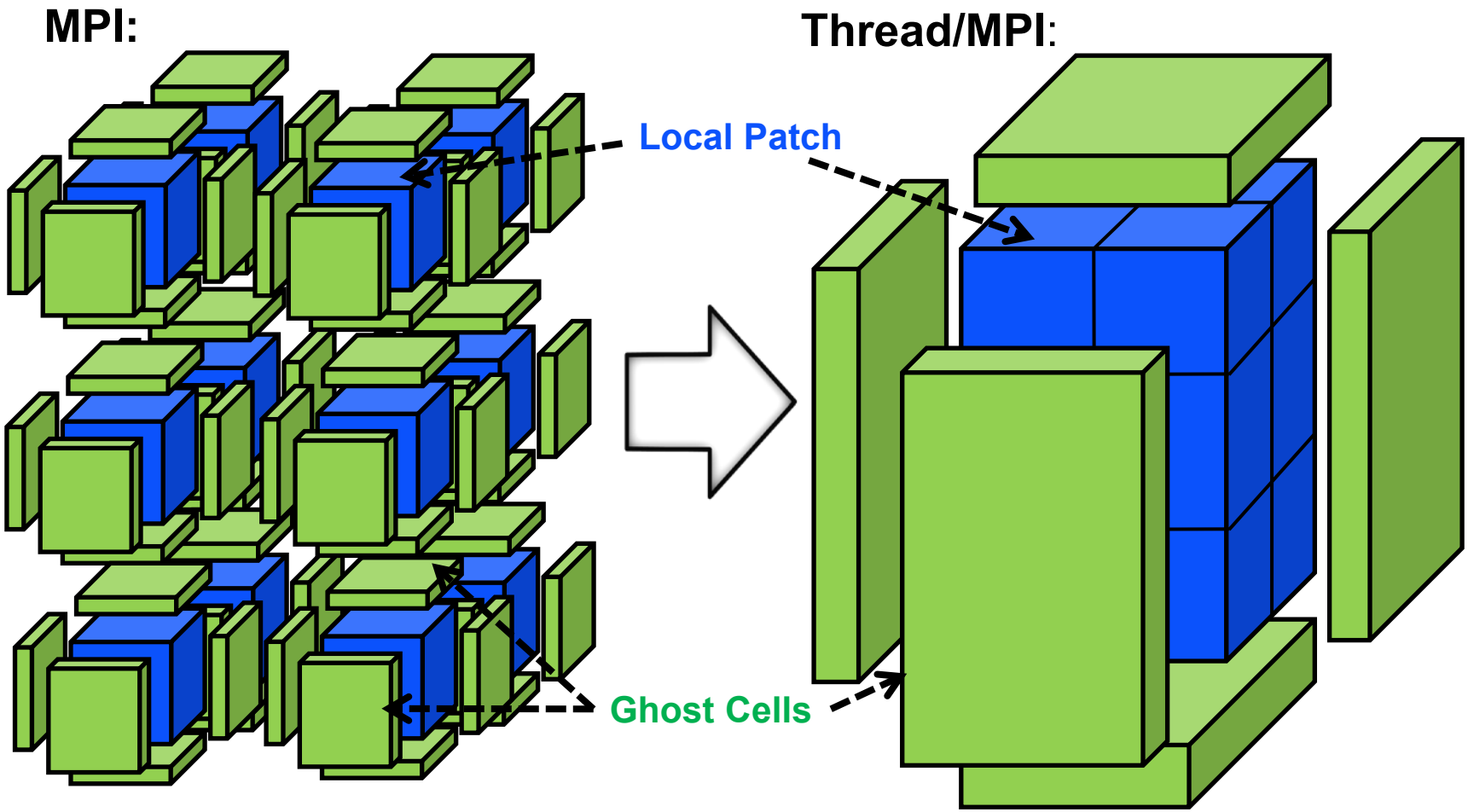


- One MPI Process per Multicore node
- All threads directly pull tasks from task queues execute tasks and process MPI sends/receives
- Tasks for one patch may run on different cores
- One data warehouse and task queue per multicore node
- Lock-free data warehouse enables all cores to access memory quickly

Uintah Runtime System



New Hybrid Model Memory Savings: Ghost Cells

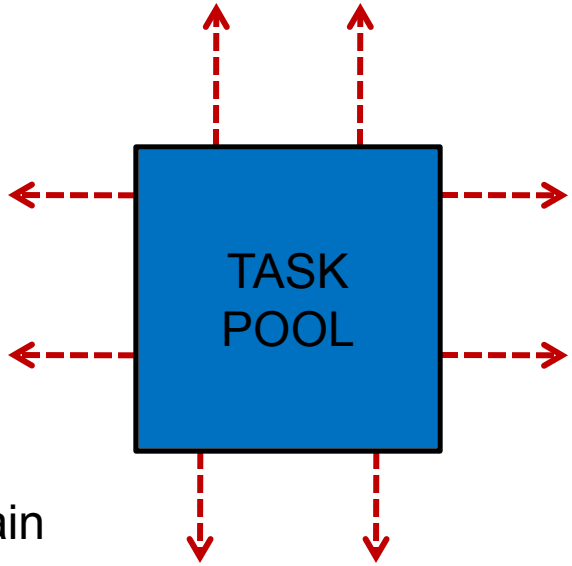


Raw Data:	49152 doubles	31360 doubles
MPI buffer:	28416 doubles	10624 doubles
Total:	75K doubles	40K doubles

(example on Kraken, 12 cores/node, 98K core 11% of memory needed

Task prioritization algorithms

Executing the task pool in different ways leads to different communications patterns



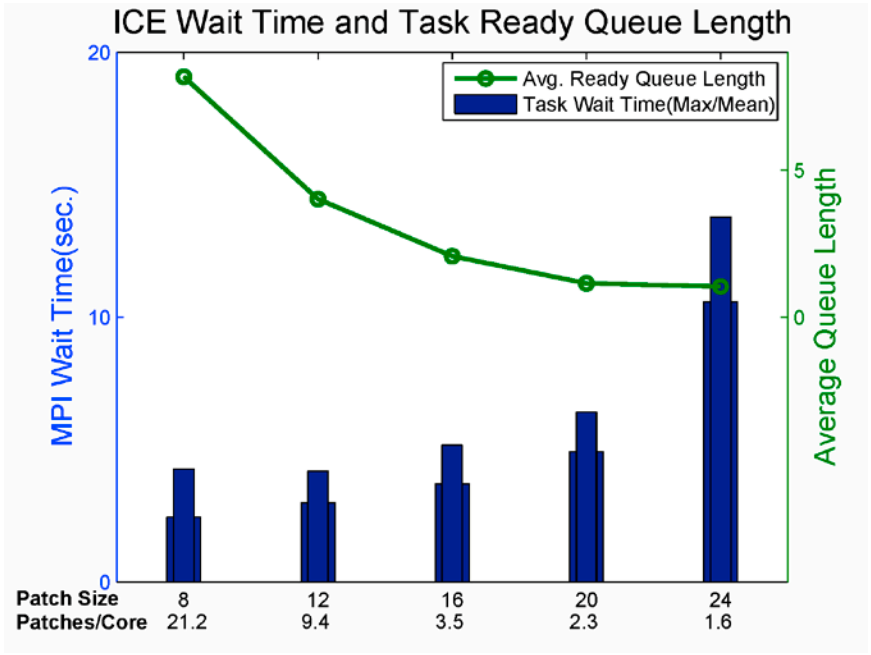
Sub-domain

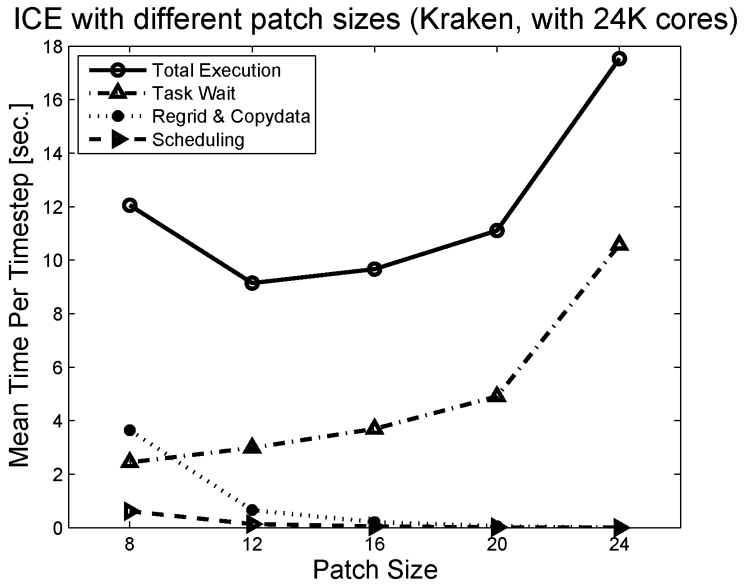
Task pool to be executed

MPI sends

Algorithm	Random	FCFS	PatchOrder	MostMsg.
Queue Length	3.11	3.16	4.05	4.29
Wait Time	18.9	18.0	7.0	2.6
Overall Time	315.35	308.73	187.19	139.39

Prioritize tasks with external communications over purely internal ones





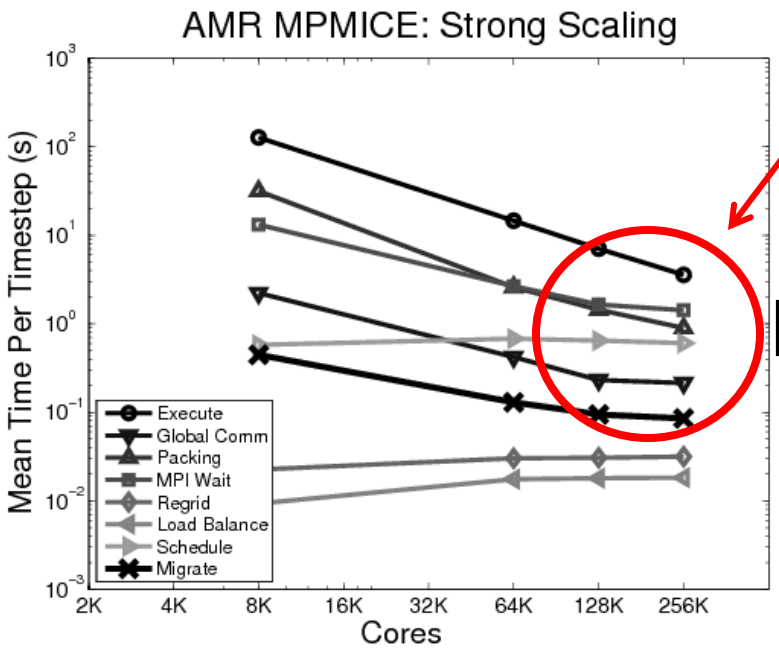
Granularity Effect

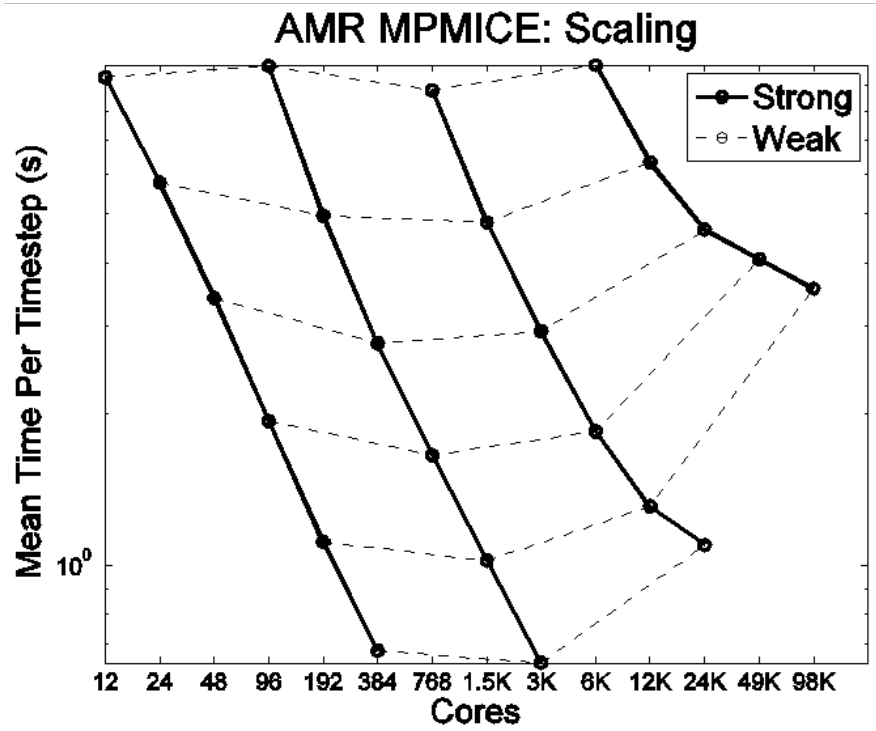
- Decrease patch size
  - (+) Increase queue length
  - (+) More overlap, lower task wait time
  - (+) More patches, better load balance
  - (-) More MPI messages
  - (-) More regrid overheads
- Other Factors
  - Problem size
  - Implied task level parallelism
  - Interconnection bandwidth and legacy
  - CPU cache size
- Solution- Self Tuning?

Nodal Performance and Global ScalbilityScalability on Titan

OLD

Scaling Breakdown





Scaling fine on Jaguar XK6 Breakdown on Jaguar XK7 with more faster cores and a faster network – needed a rewrite of Data Warehouse to allow cores faster access

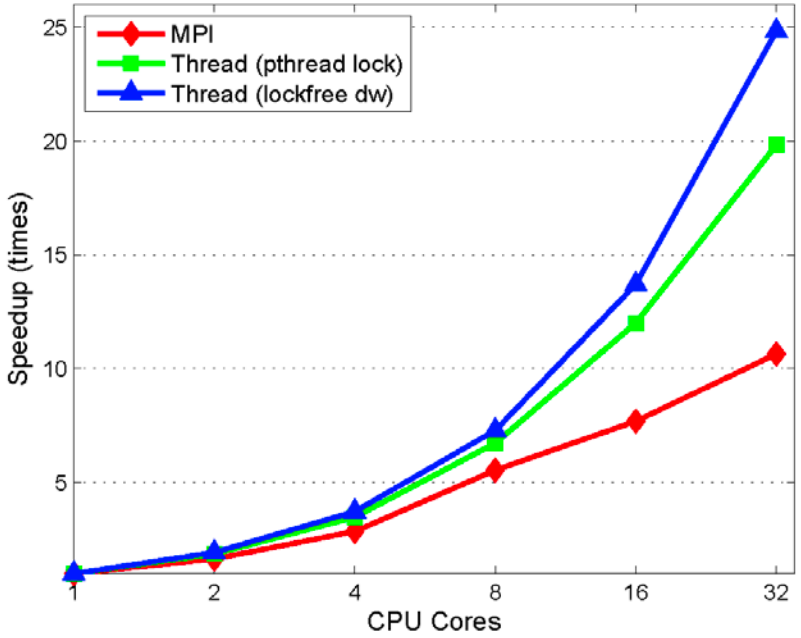
One flow with particles moving
3-level AMR MPM ICE 70% efficiency
At 256K cores vs 16K cores

THE UNIVERSITY OF UTAH

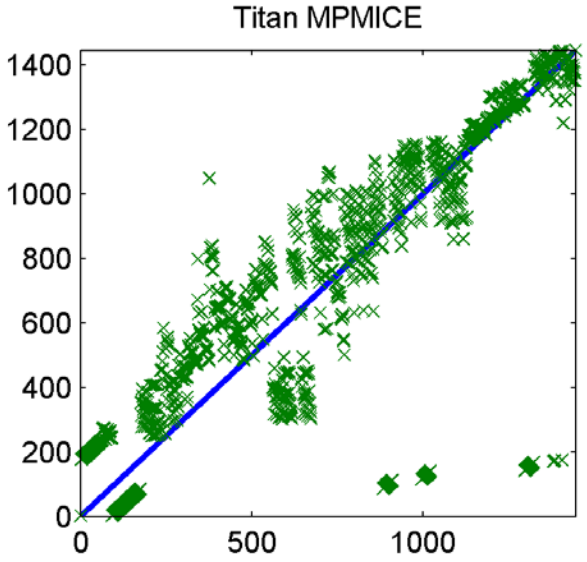
Lock-Free Data Structures

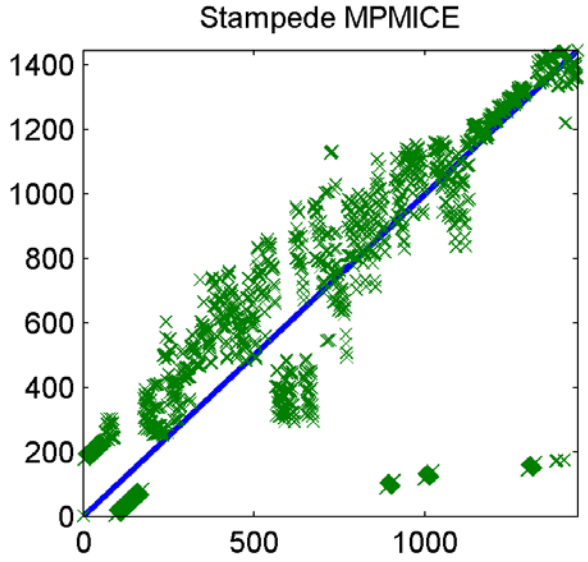
Global scalability depends on the details of nodal run-time system. Change from Jaguar to Titan – more faster cores and faster communications broke our Runtime System which worked fine with locks previously

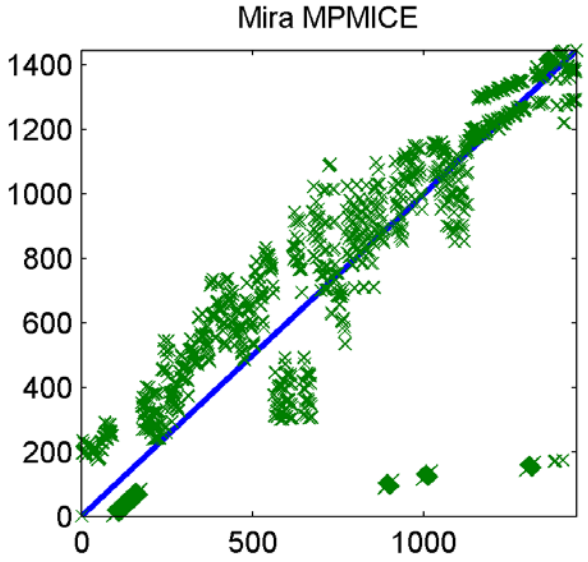
- Using atomic instruction set
- Variable reference counting
  - *fetch_and_add, fetch_and_sub*
    compare_and_swap
  - *both read and write simultaneously*
- Data warehouse
  - Redesigned variable container
  - Update: *compare_and_swap*
  - Reduce: *test_and_set*



Scalability is at least partially achieved by not executing tasks in order e.g. AMR fluid-structure interaction





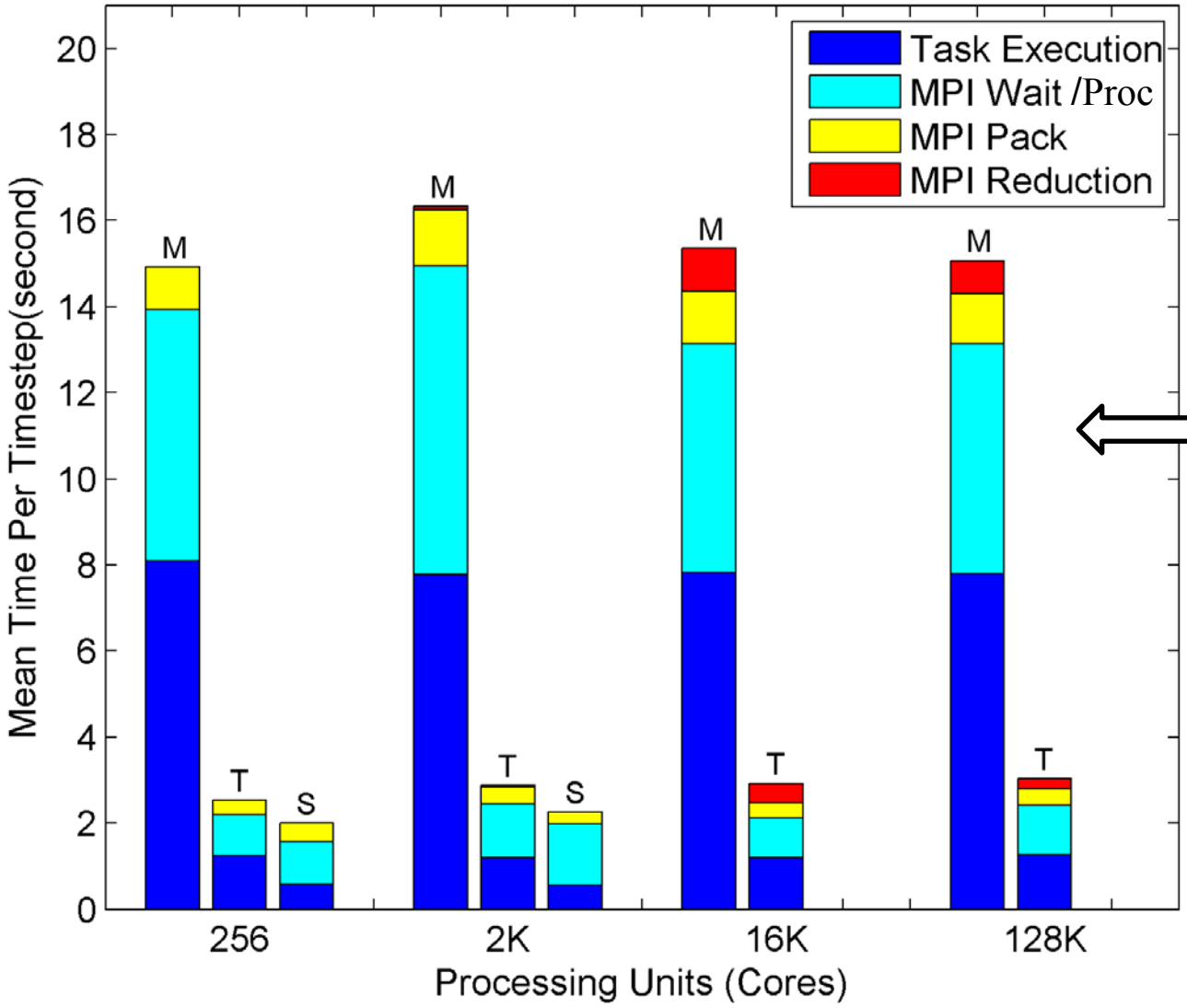


Straight line represents given order of tasks Green X shows when a task is actually executed.
Above the line means late execution while below the line means early execution took place. More "late" tasks than "early" ones as e.g.

TASKS: 1 2 3 4 5 ⇨ 1 4 2 3 5

Early Late execution

Weak Scaling AMR+MPM ICE
M = Mira, T=Titan, S=Stampede



Only 2 patches per core Includes packing unpacking and data warehouse

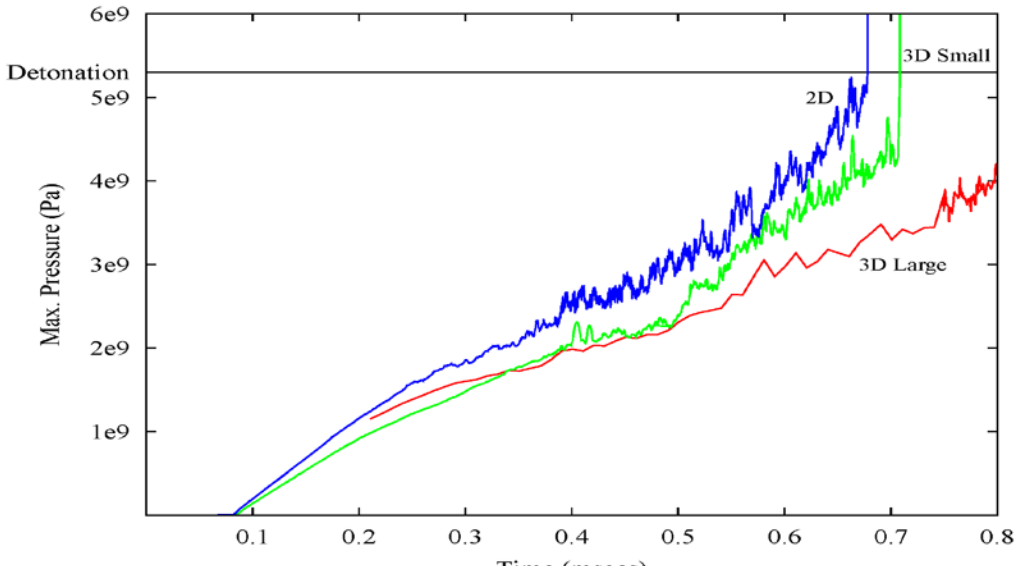
Only 8 interior patches from 32

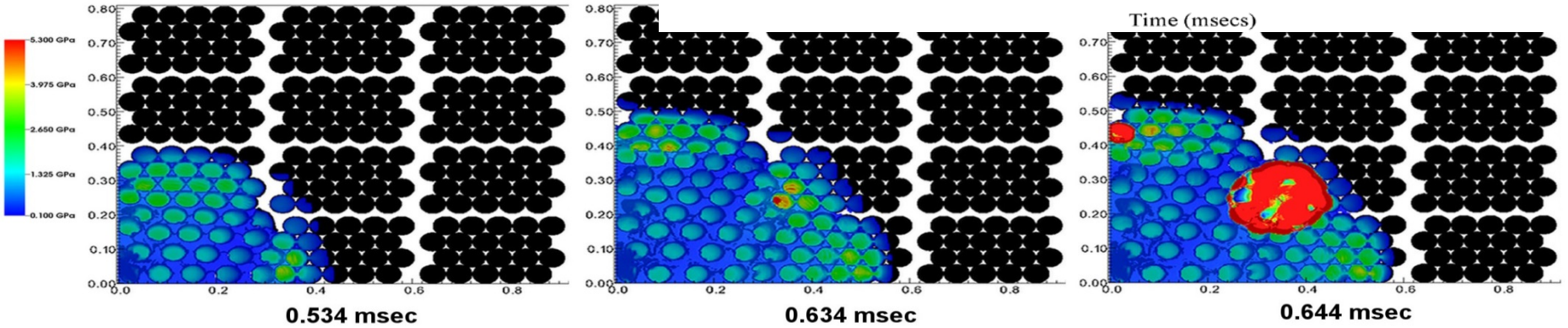
NSF funded modeling of Spanish Fork Accident 8/10/05

Speeding truck with 8000 explosive boosters each with 2.5-5.5 lbs of explosive overturned and caught fire

Experimental evidence for a transition from deflagration to detonation?

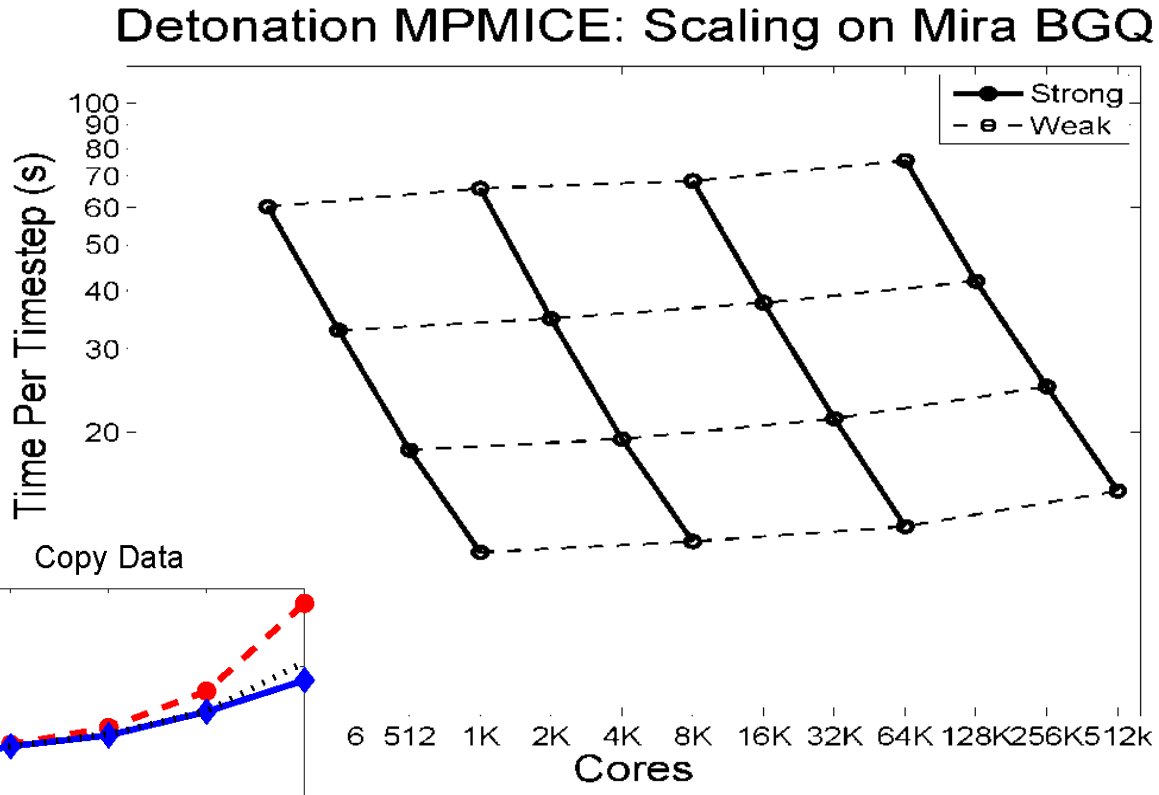
Deflagration wave moves at ~400m/s not all explosive consumed. Detonation wave moves 8500m/s all explosive consumed.

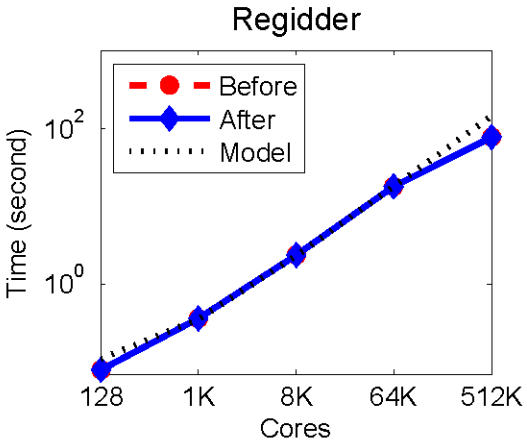


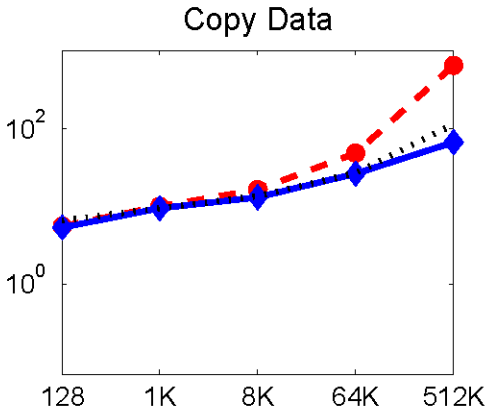


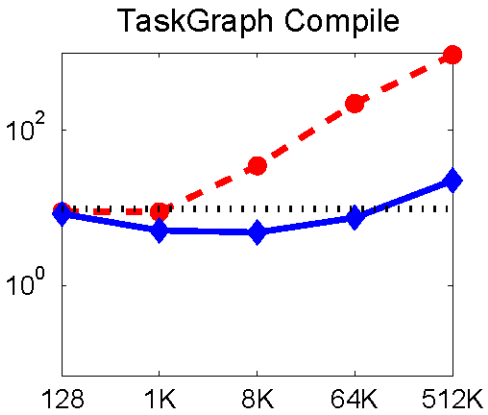
Spanish Fork Accident

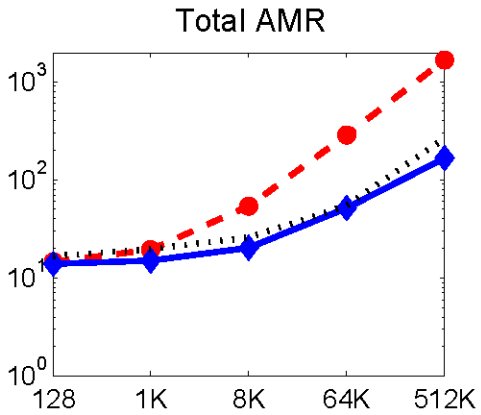
500K mesh patches
1.3 Billion mesh cells
7.8 Billion particles







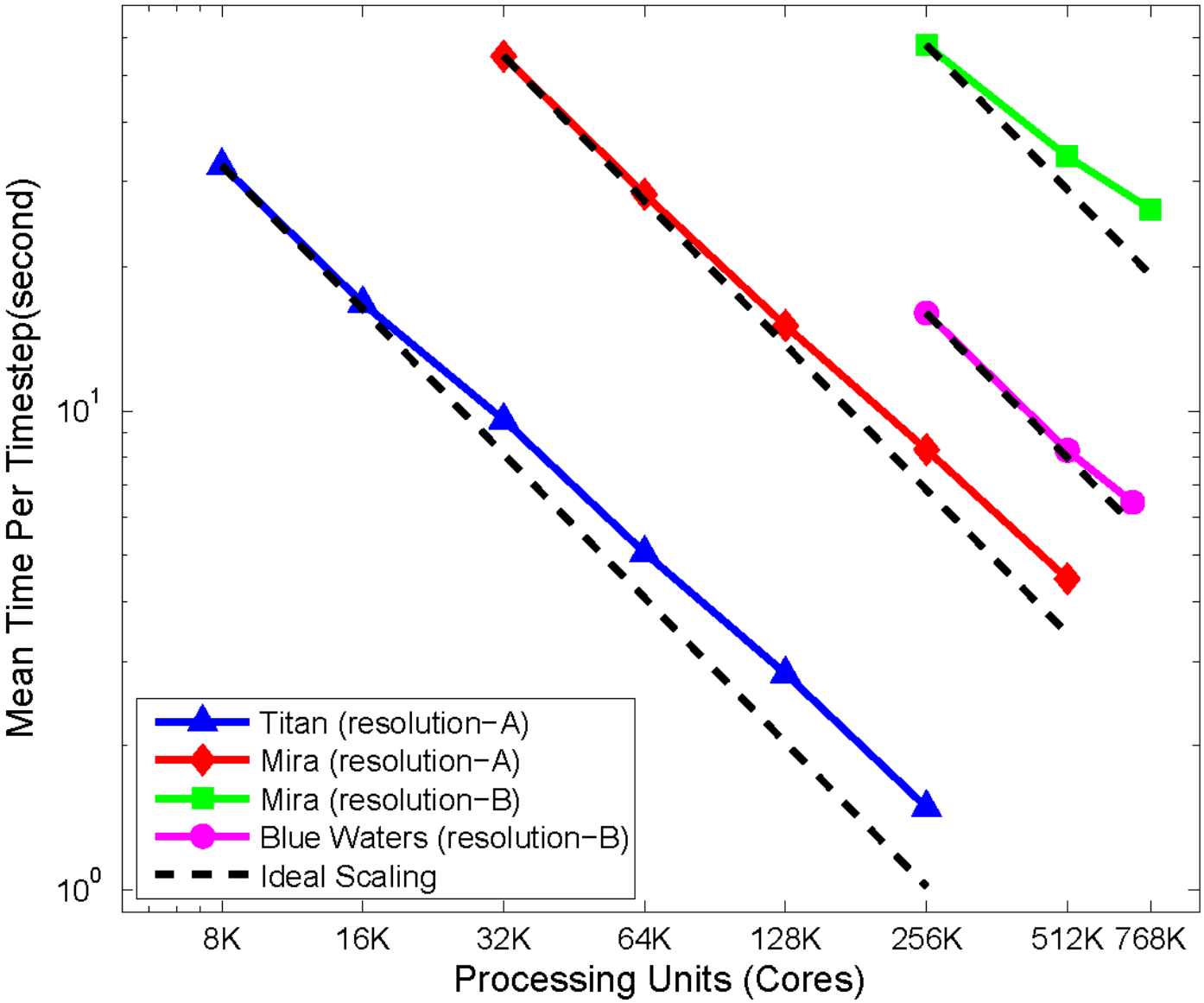




At every stage when we move to the next generation of problems Some of the algorithms and data structures need to be replaced .

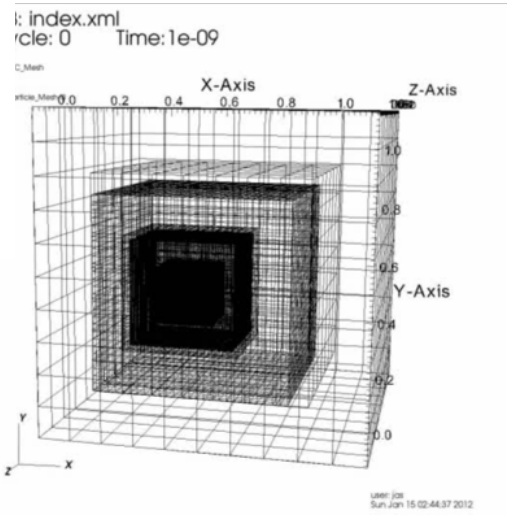
Scalability at one level is no certain Indicator fro problems or machines An order of magnitude larger

MPM AMR ICE Strong Scaling



Mira DOE BG/Q
768K cores
Blue Waters Cray XE6/XK7 700K+ cores

Resolution B
29 Billion particles
4 Billion mesh cells
1.2 Million mesh patches



Complex fluid-structure interaction problem with adaptive mesh refinement, see SC13/14 paper

NSF funding.

Summary of Scalability Improvements

(i) Move to a one MPI process per multicore node reduces memory to less than 10% of previous for 100K+ cores

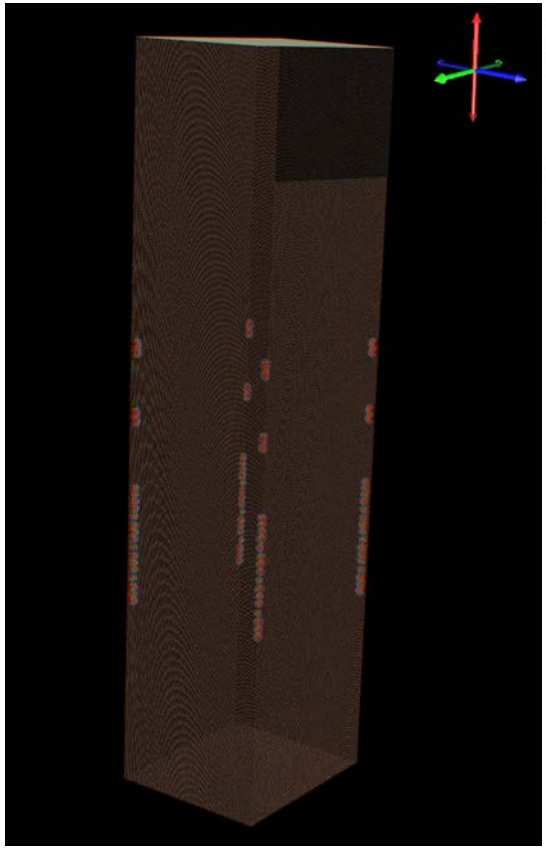
(ii) Use optimal size patches to balance overhead and granularity 16x16x 16 to 30x30x30.

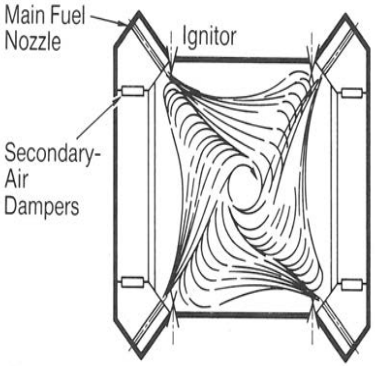
(iii) Use only one data warehouse but allow all cores fast access to it, through the use of atomic operations.

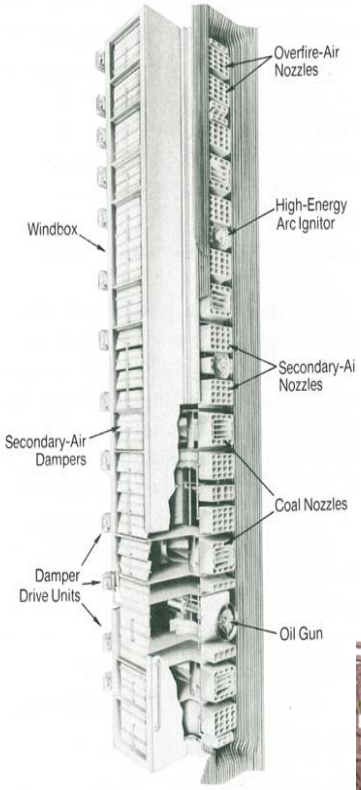
(iv) Prioritize tasks with the most external communications

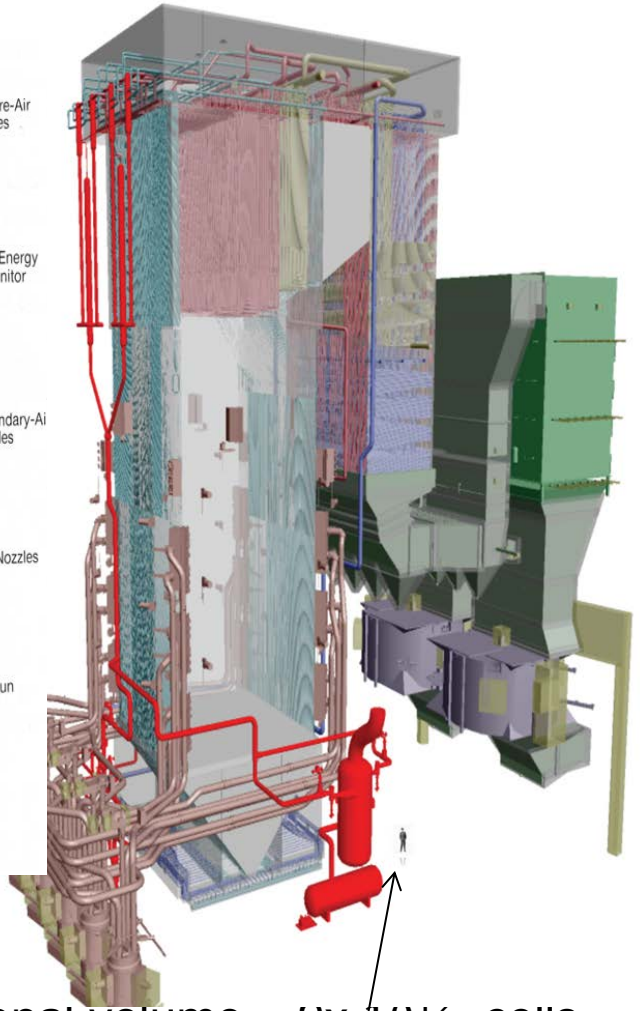
(v) Use out-of-order execution when possible

An Exascale Design Problem - Alstom Clean Coal Boilers









For 350MWe boiler problem. LES resolu
needed: 1mm per side for each computational volume = 9x 10^{12} cells
This is one thousand times larger than the largest problems we solve today.

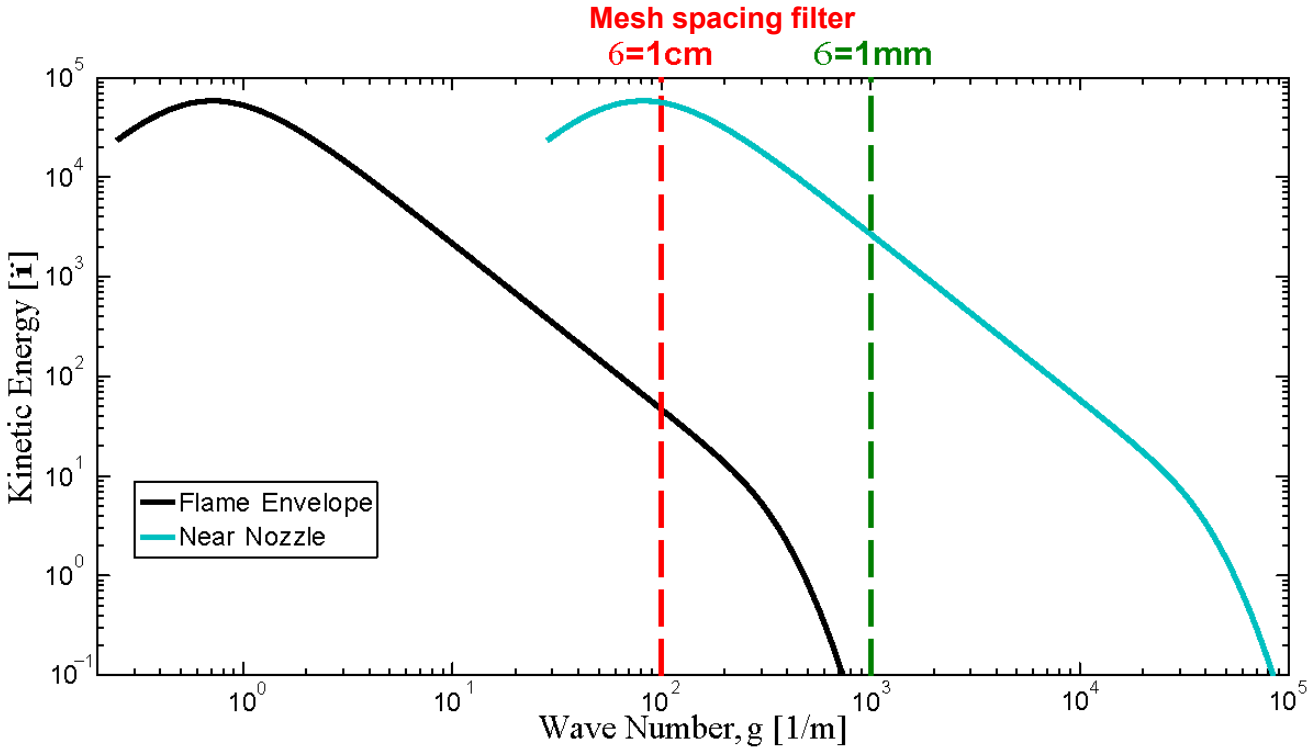
Prof. Phil Smith Dr Jeremy Thornock ICSE

Existing Simulations of Boilers using ARCHES in Uintah

(i) Traditional Lagrangian/RANS approaches do not address well particle effects

(ii) LES has potential to predict oxy---coal flames and to be an important design tool

(iii) LES is "like DNS" for coal, but 1mm mesh needed to capture phenomena

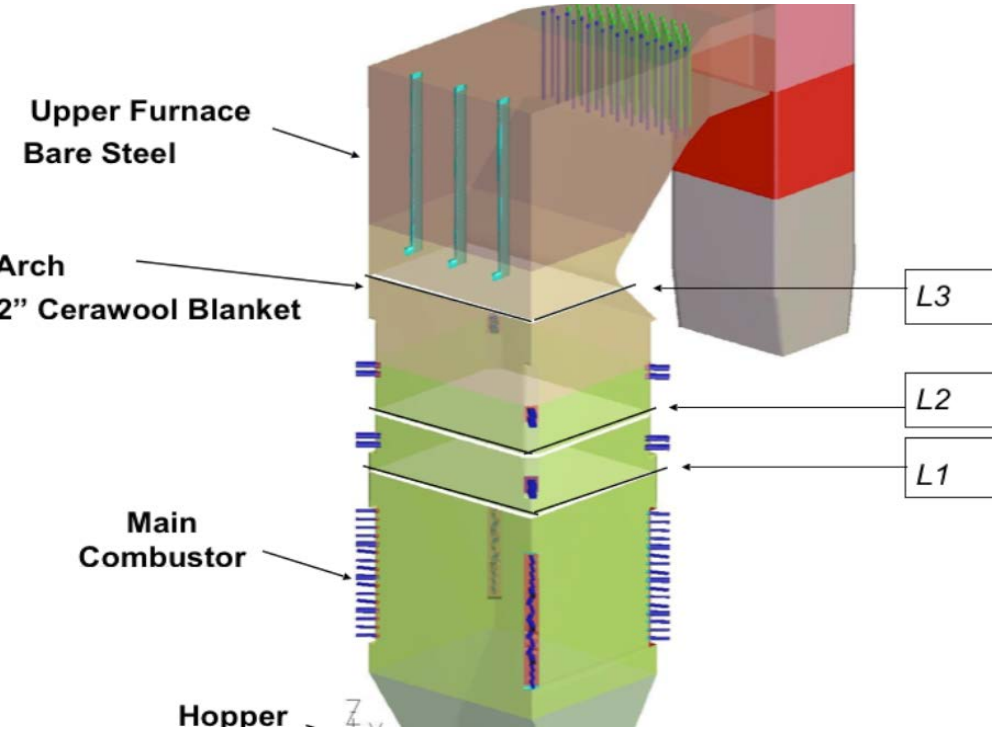


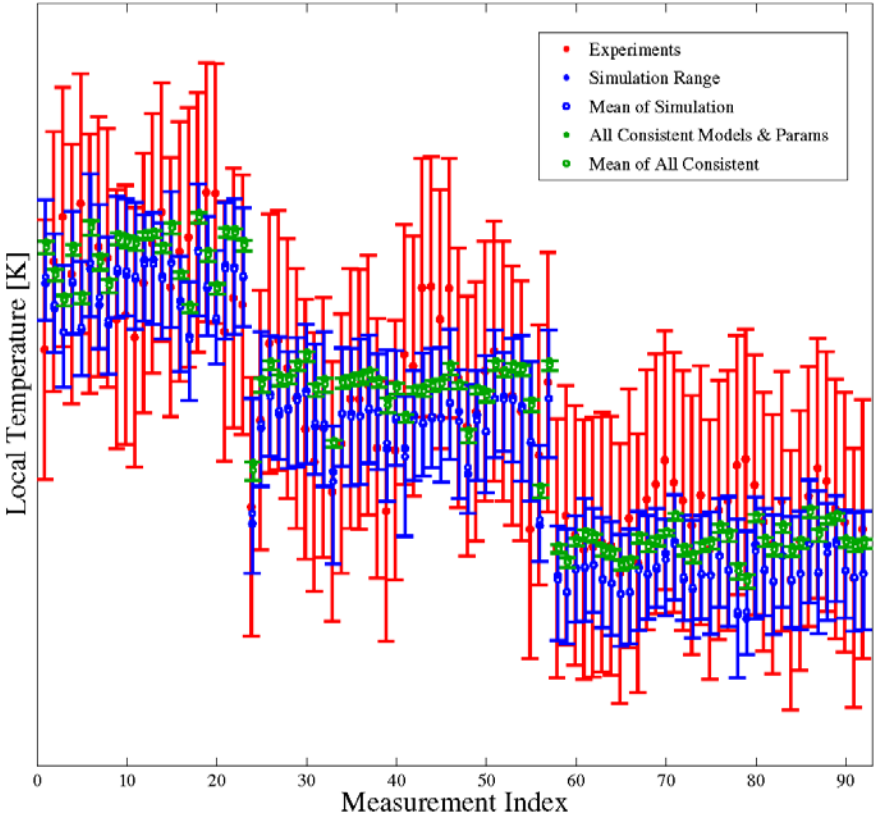
Structured, finite-volume method, Mass, momentum, energy with radiation

Higher-order temporal/spatial numerics, LES closure, Tabulated chemistry

PDF mixing models, DQMOM, modeling particles

Uncertainty Quantified Runs on a Small Prototype Boiler



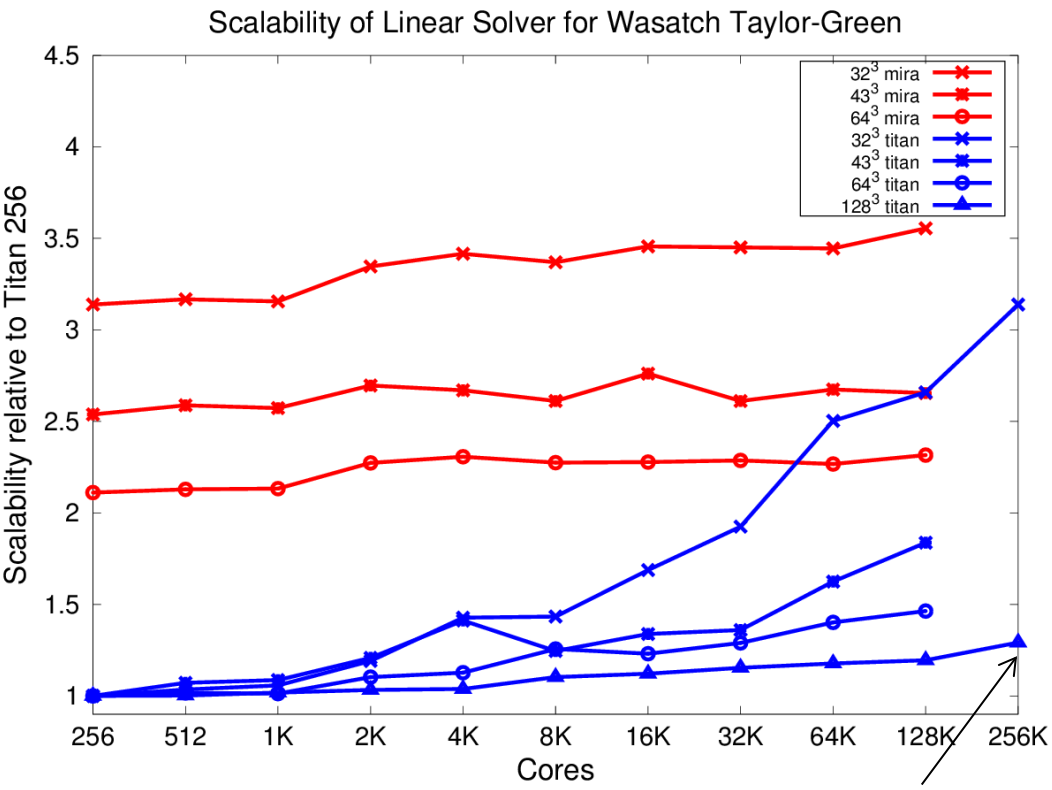


Red is experiment
Blue is simulation
Green is consistent

Absence of scales for commercial reasons

[Source: Jeremy Thornock]

Linear Solves arises from Low Mach Number Navier –Stokes Equations

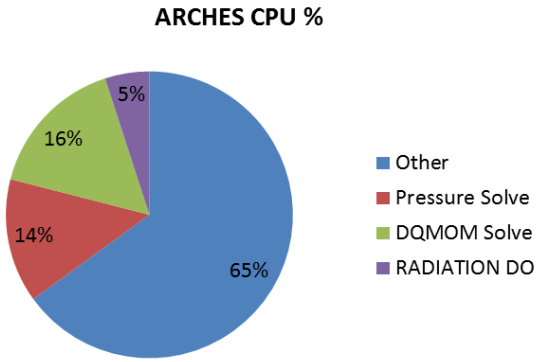


$$\nabla^2 p = R, \quad \text{where } R = \nabla . F + \frac{\partial^2 p}{\partial t^2}$$

Use Hypre Solver from LLNL Preconditioned Conjugate Gradients on regular mesh patches used

Multi-grid pre-conditioner used
Careful adaptive strategies needed to get scalability

2.2 Trillion DOF



Each **Mira Run** is scaled wrt the **Titan Run at 256 cores**
Note these times are not the same for different patch sizes.

Weak Scalability of Hypre Code

One radiation solve every 10 timesteps

NEBO/Wasatch Example

- **Express complex pde functions as DAG** - automatically construct algorithms from expressions
- **Define field operations** needed to execute tasks (fine grained vector parallelism on the mesh)
- **User writes only field operations code** . Supports field & stencil operations directly - no more loops!
- **Strongly typed fields** ensure valid operations at compile time. *Allows a variety of implementations to be tried without modifying application code.*
- **Scalability on a node** - use **Uintah** infrastructure to get scalability across whole system

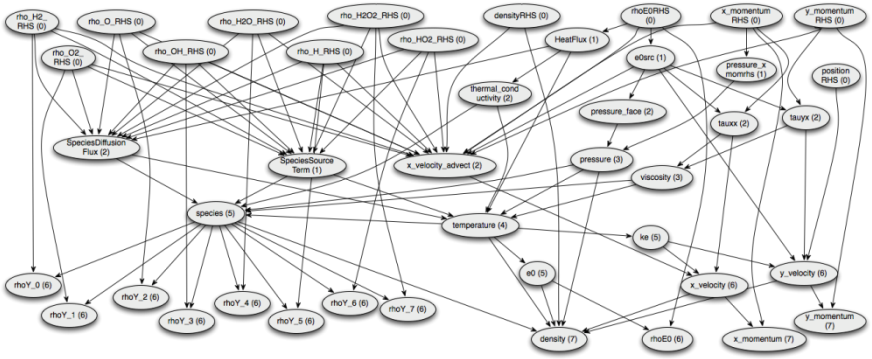
Energy equation

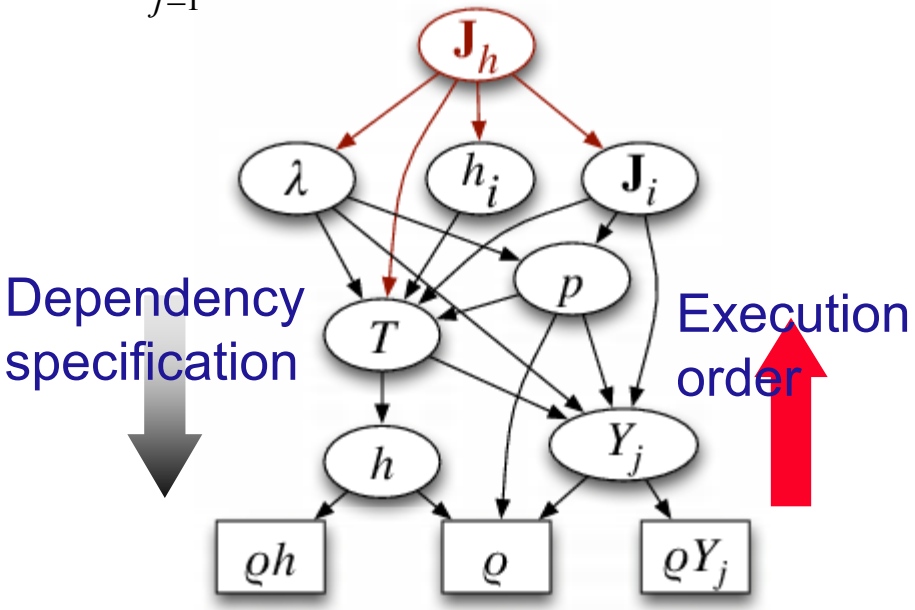
$$\frac{\partial \rho e}{\partial t} + \nabla.(\rho e \underline{u}) + \nabla.\underline{J}_h + terms = 0$$

Enthalpy diffusive flux

$$\underline{J}_h = -\lambda(T, Y_j)\nabla T - \sum_{i=1}^{n} h_i \underline{J}_i$$

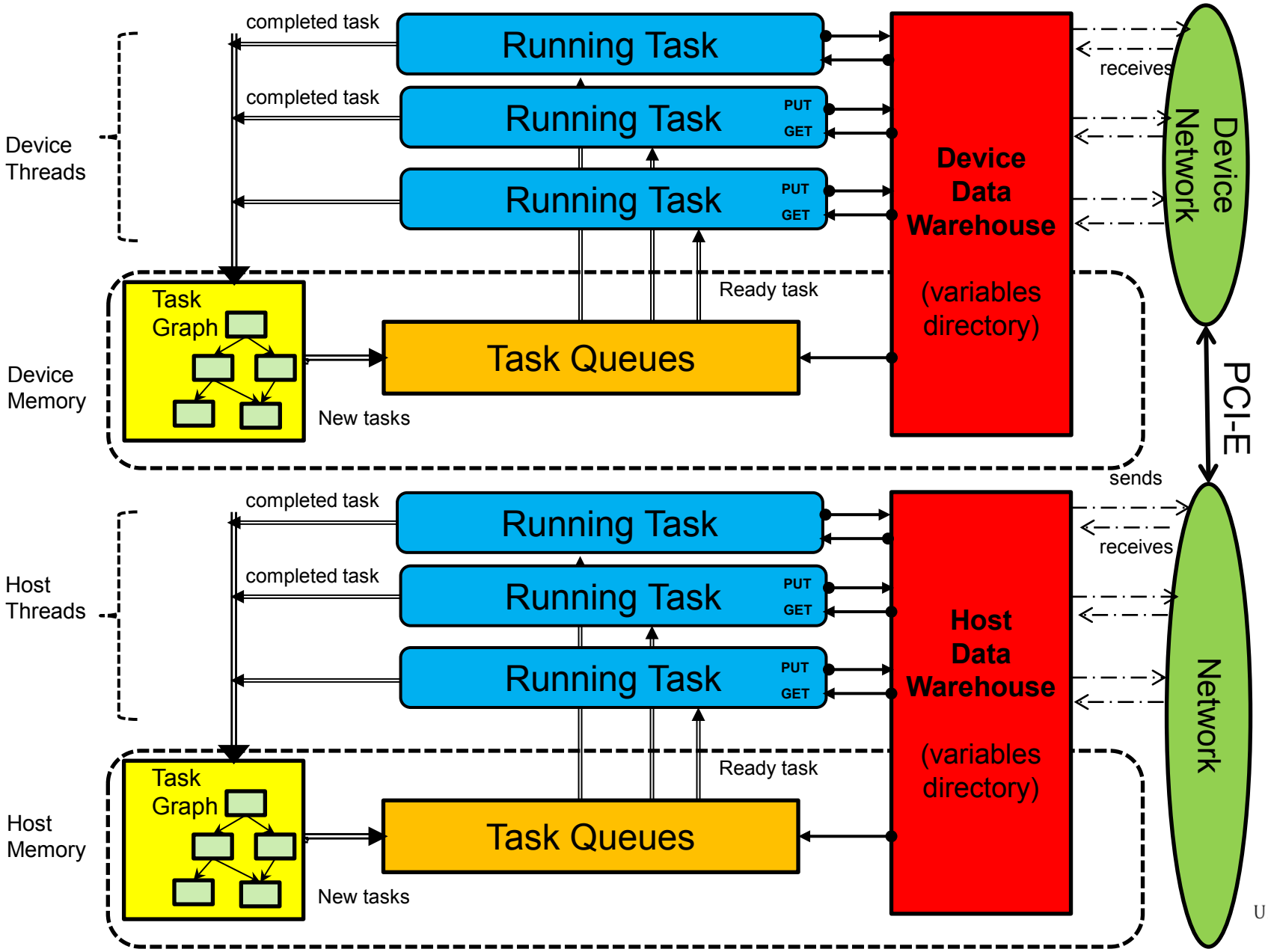
$$\underline{J}_i = -\sum_{j=1}^{ns} D_{ij}(T, Y_j)\nabla Y_j - D_i^T(T, Y_j)\nabla T$$





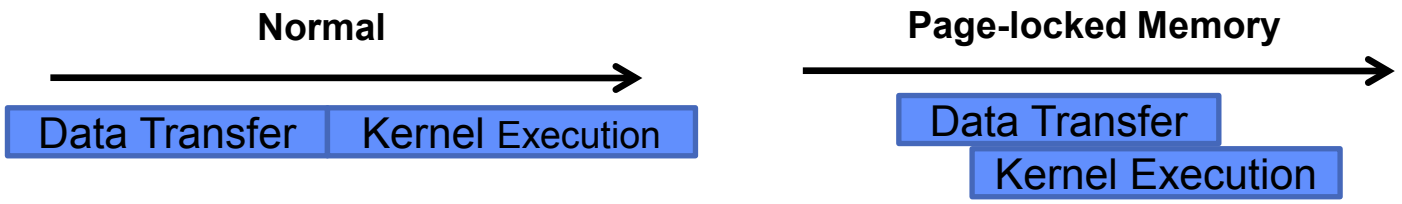
[Sutherland Earl Might]

Unified Heterogeneous Scheduler (GPU or Phi symmetric)

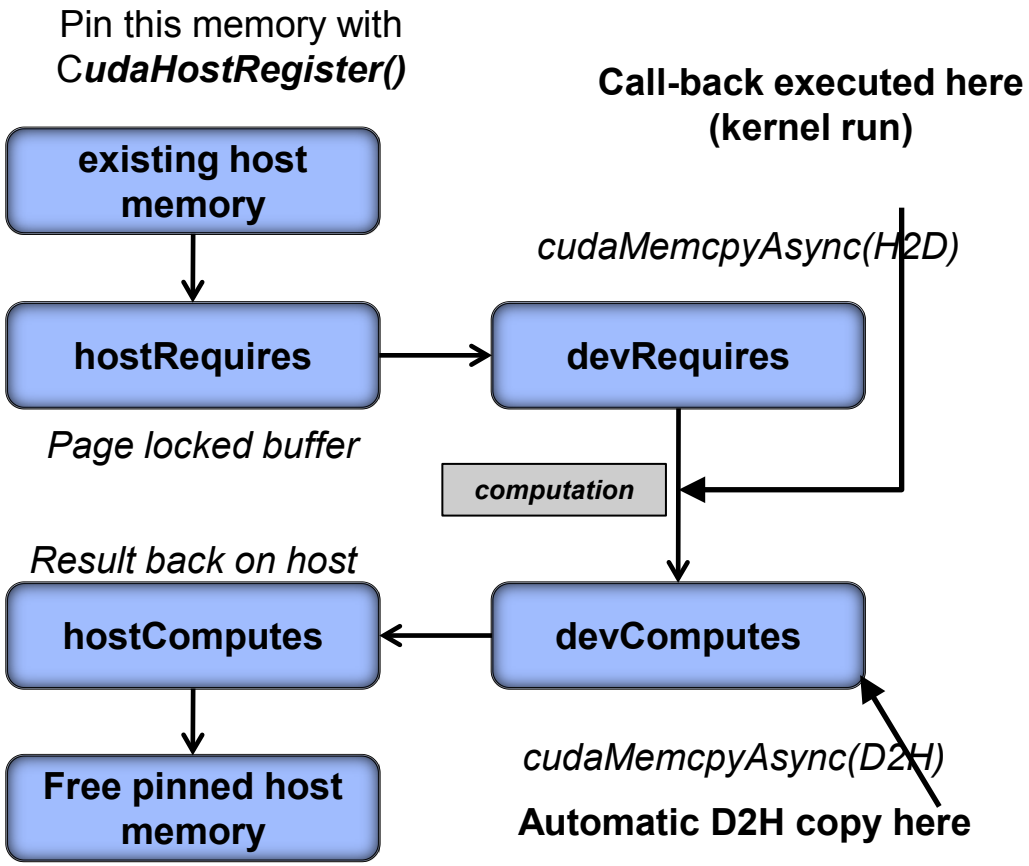


GPU Task and Data Management

Framework Manages Data Movement
Host ← → Device

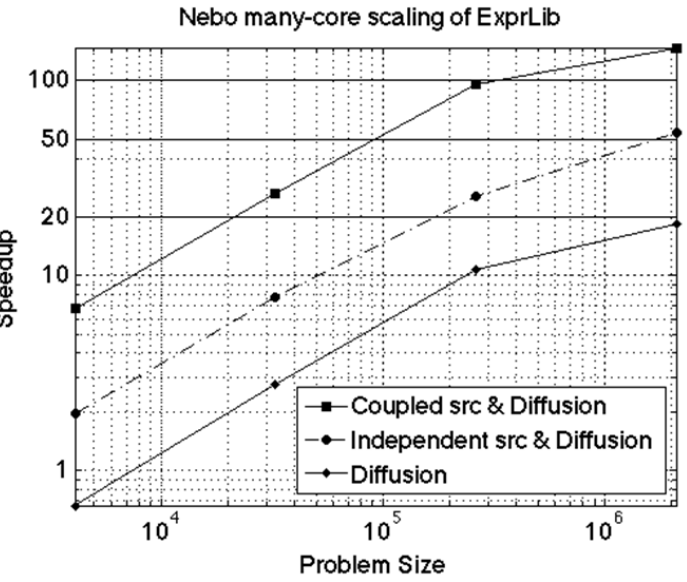


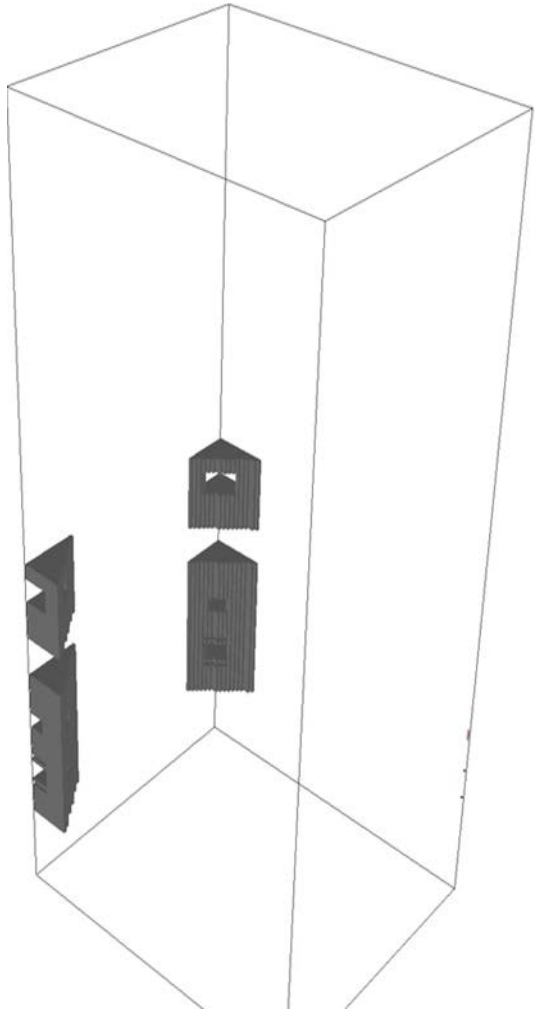
- Use ***CUDA Asynchronous API***
- **Automatically** generate CUDA streams for task dependencies
- **Concurrently** execute kernels and memory copies
- **Preload** data before task kernel executes
- **Multi-GPU** support



Wasatch – Nebo Recent Milestones

- Wasatch is solving (nonreacting miniboiler~3-4x speedup over the non-DSL approach.
- New Nebo backend for CPU resultied in 20-30% speedup in the entire Wasatch code base.
- Much of the Wasatch code base is GPU-ready
- Arches plus SpatialOps & Nebo EDSL being scoped.





Good GPU scaling with (>32^3 per patch).
Loop fusion (heavy GPU kernels) needed e.g “coupled source & diffusion”

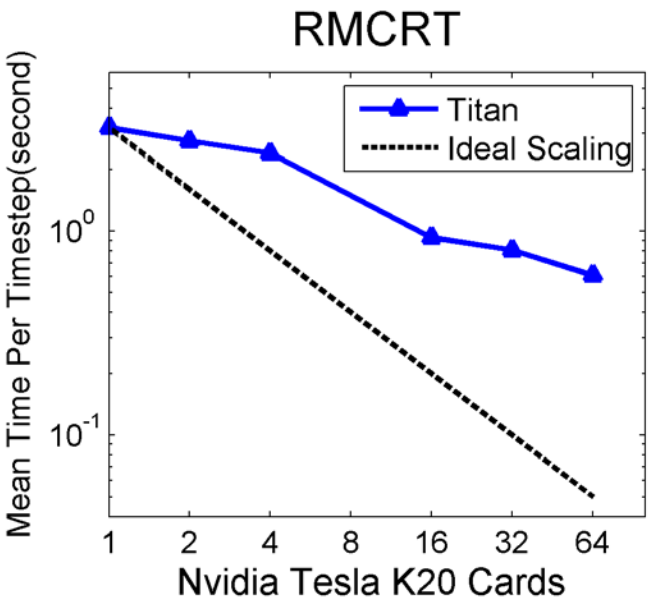
[James Sutherland]

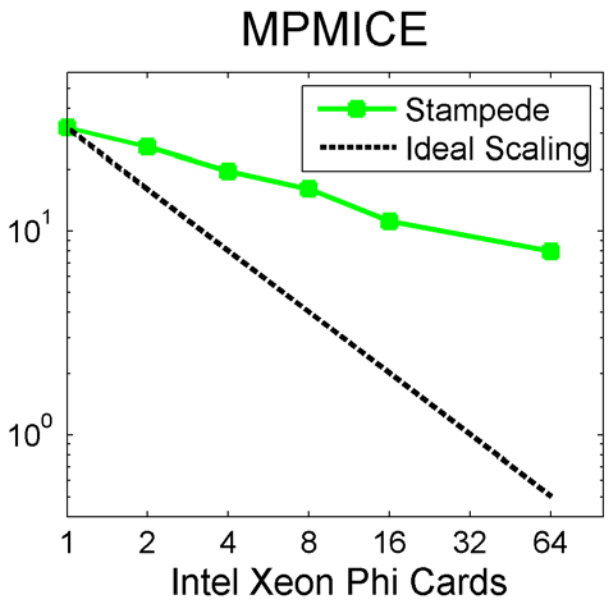
DESIGNING FOR EXASCALE

Clear trend towards accelerators e.g. GPU but also Intel MIC – new NSF "Stampede" 10-. 15PF Balance factor = flops/bandwidth - high

GPU performance "ok" for stencil-based codes ,2x over multicore cpu estimated and achieved for ICE . Similar results by others.
Network and memory performance more slowly growing than cpu/gpu performance. GPU perf.of ray-tracing radiation method is 100x cpu

Overlapping and hiding Communications essential





NVIDIA AMGX Linear Solvers on GPUs

- Fast, scalable iterative gpu linear solvers for packages e.g.,
- Flexible toolkit provides GPU accelerated Ax = b solver
- Simple API for multiple apps domains.
- Multiple GPUs (maybe thousands) with scaling

Key Features

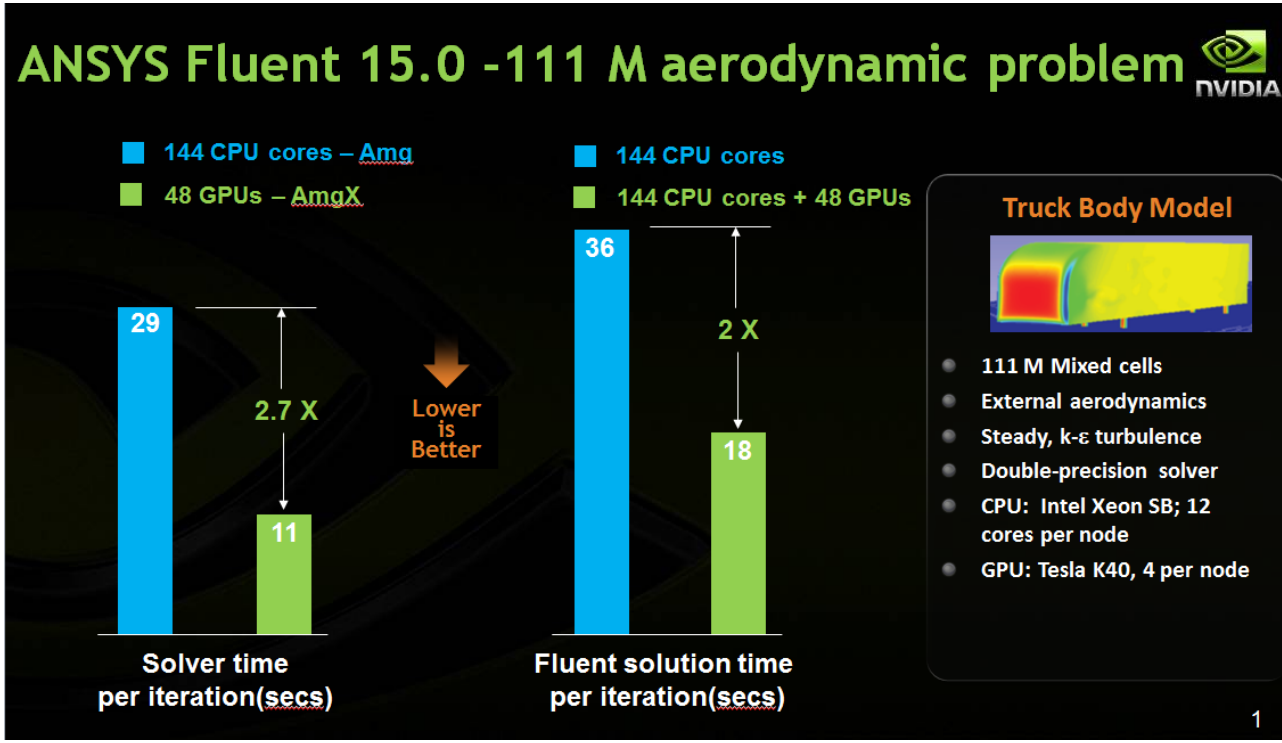
Ruge-Steuben algebraic MG

Krylov methods: CG, GMRES, BiCGStab,

Smoothers and Solvers: Block- Jacobi, Gauss-Seidel, incomplete LU,

Flexible composition system

MPI support OpenMP support, Flexible and high level C API,



Free for non-commercial use

Utah access via Utah CUDA COE.

DESIGNING FOR EXASCALE

Clear trend towards accelerators e.g. GPU but also Intel MIC – NSF "Stampede" Balance factor = flops/bandwidth – high.PORTABILITY IS THE KEY ISSUE:NEW CODE - use Wasatch to generate code for GPUs and MICs .How do we handle the challenge of existing code?

Kokkos: A Layered Collection of Libraries

- **Standard C++, Not a language extension**
  - **In *spirit* of TBB, Thrust & CUSP, C++AMP, LLNL's RAJA, ...**
  - ***Not* a language extension like OpenMP, OpenACC, OpenCL, CUDA, ...**
- **Uses C++ template meta-programming**
- **Multidimensional Arrays, *with a twist***
  - **Layout mapping: multi-index (i,j,k,...) ↔ memory location**
  - ➢ **Choose layout to satisfy device-specific memory access pattern**
  - **Layout changes are invisible to the user code**

[source Carter Edwards and Dan Sunderland]

Evaluate Performance Impact of Array Layout [Edwards and Sunderland]

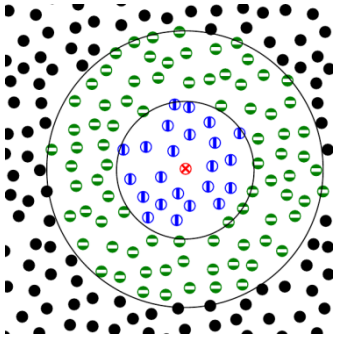
Sandia National Laboratories

- Molecular dynamics computational kernel in miniMD
- Simple Lennard Jones force model:

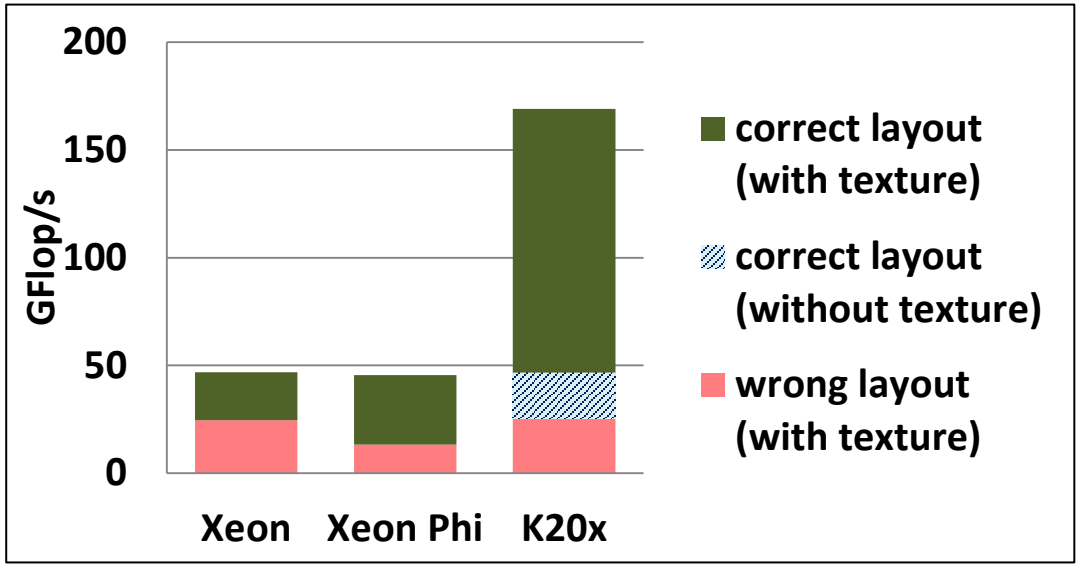
$$F_i = \sum_{j, r_{ij} < r_{cut}} 6\varepsilon \left[\left(\frac{\varsigma}{r_{ij}} \right)^7 - 2 \left(\frac{\varsigma}{r_{ij}} \right)^{13} \right]$$

- Atom neighbor list to avoid N^2 computations

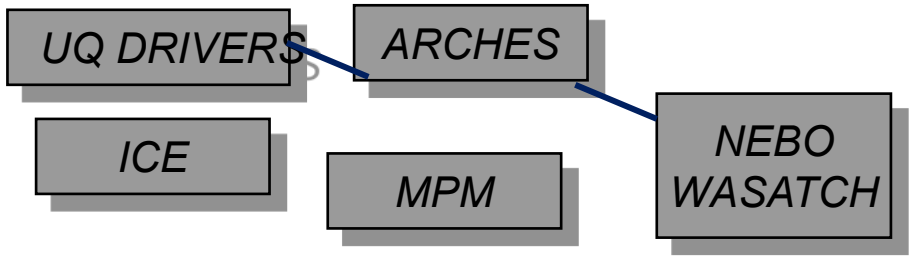
```
pos_i = pos(i);
for( jj = 0; jj < num_neighbors(i); jj++) {
  j = neighbors(i,jj);
  r_ij = pos_i - pos(j); //random read 3 floats for pos
  if (|r_ij| < r_cut) f_i += 6*e*((s/r_ij)^7 - 2*(s/r_ij)^13)
}
f(i) = f_i;
```



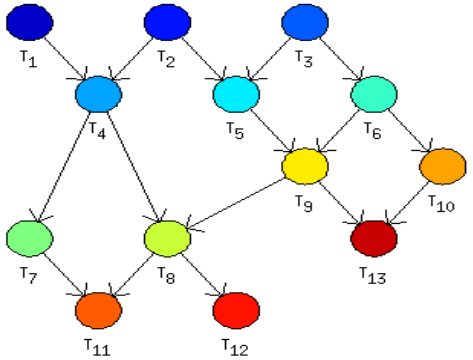
- Test Problem
  - 864k atoms, ~77 neighbors
  - 2D neighbor array
  - Different layouts CPU vs GPU
  - Random read 'pos' through GPU texture cache
- Large performance loss with wrong array layout



Applications code

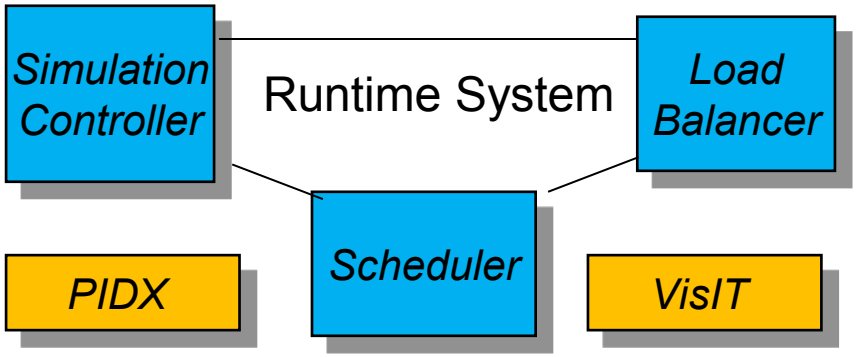


Abstract C++ Task Graph Form



Compilation into C++ Cuda etc

Adaptive Execution of tasks



On specific processors

Proposed Uintah(X) Architecture Decomposition

Resilience

- **Need interfaces at system level to help us consider:**
- Core failure – reroute tasks
- Comms failure – reroute message
- Node failure – need to replicate patches use an AMR type approach in which a coarse patch is on another node. In 3D has 12.5% overhead – suggested by Qingyu Meng Mike Heroux and others.
- **Will explore this from fall 2014 onwards. Just how bad is the problem?**

Summary

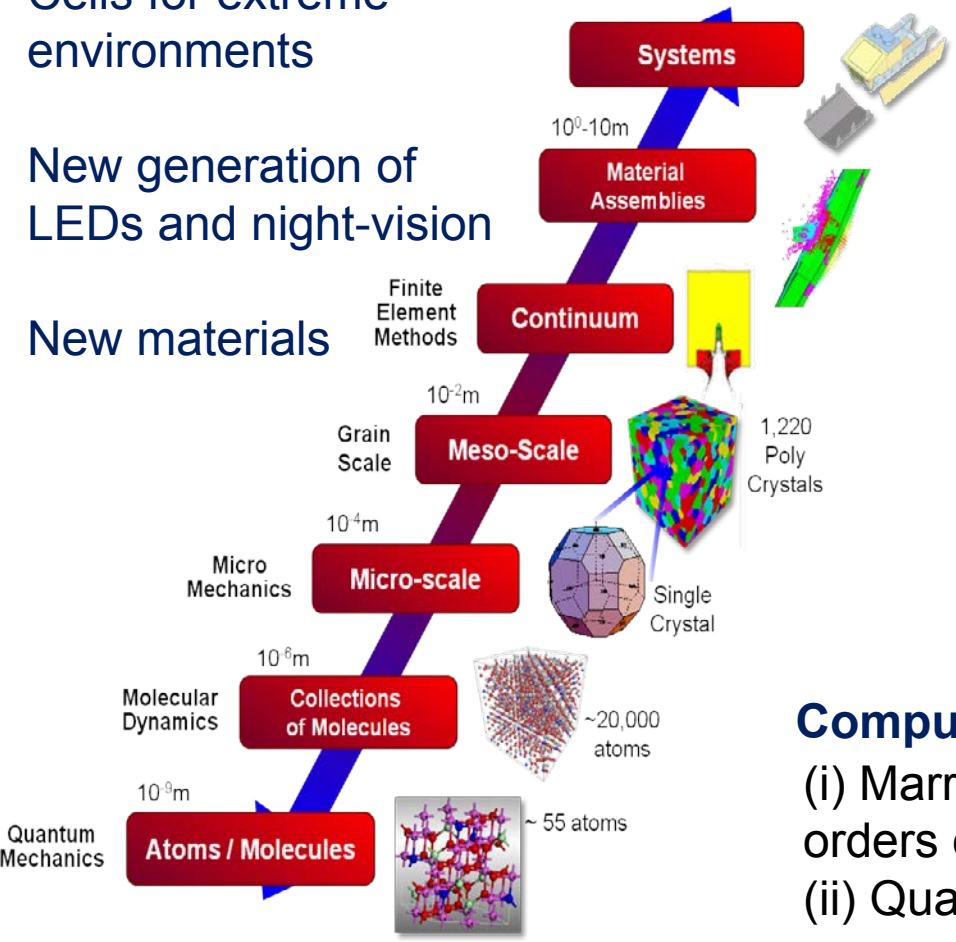
- DAG abstraction important for achieving scaling
- Layered approach very important for not needing to change applications code
- Scalability still requires much engineering of the runtime system.
- General approach very powerful indeed.
- Obvious applicability to new architectures
- DSL approach very important very future
- Scalability still a challenge even with DAG approach – which does work amazingly well, e.g. for fluid-structure calculations
- GPU and MIC development ongoing
- The approach used here shows promise for very large core and GPU counts but using these architectures is an exciting challenge

ARL: Multi-scale Modeling of Electronic Materials
Utah, Boston, RPI, Chicago, Harvard, Brown, Penn State

Vision: Longer lasting batteries and fuel Cells for extreme environments

New generation of LEDs and night-vision

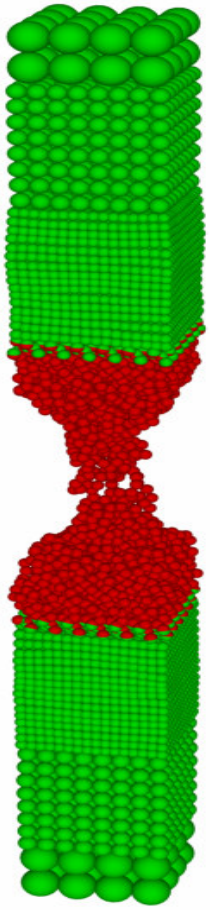
New materials



- **Complex problems require differing scales**

Example: Battery Cathode (Atomistic/CG + MPM)

Mesoscopic (or larger) cathode particle mechanical response via MPM.Microscopic particle/electrolyte interactions a Atomistic/CG scale



Example of AMR MPM Coupling with MD [Nitin Daphalapurkar]

Computational Challenges

(i) Marrying simulation techniques across multiple orders of magnitudes.

(ii) Quantifying Uncertainty across multiple scales

Weak and Strong Scalability:

Problem size n on p cores takes time T(n,p)

Strong Scalability $T(n,p) = T(n,1) / p$

Try to solve the same problem p times more quickly on p cores

Weak Scalability $T(np,p) = T(n,1)$

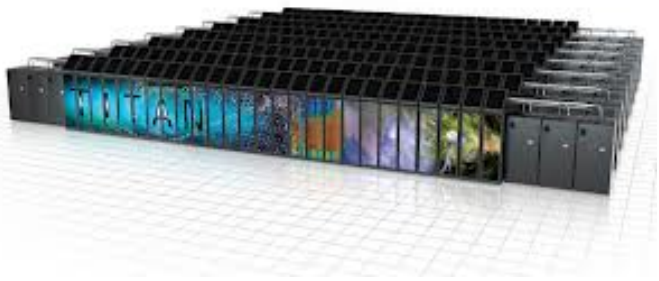
Solve a problem that is p times as large in the same time on p cores

Theorem

Both weak and strong scalability only if **linear complexity**

[Tirado + Martin] 1998

$$T(n,1) = \alpha n$$



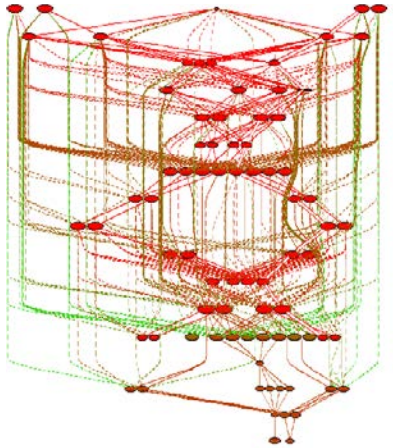
Today's machines used in this talk

SYSTEM	Vendor/ Type	CPUs and Accelerators	Cores	Mem/ Node	Inter-conn.	Peak Pflop
TITAN	Cray XK7	AMD Opteron 2.6Ghz NVIDIA KEPLER	299008 18K x 2496	32GB	Cray Gemini	27
Stampede	Dell Zeus	Intel Sandybridge 2,7GHz Intel Xeon Phi	102400 390400	32GB	Infinib-and	4
Mira	IBM Blue Gene Q	Power PC A2 1.6Ghz	786432	16GB	5D Torus	10

NSFs Kraken and DOEs Titan, DoD machines and local HP machines are our workhorses.

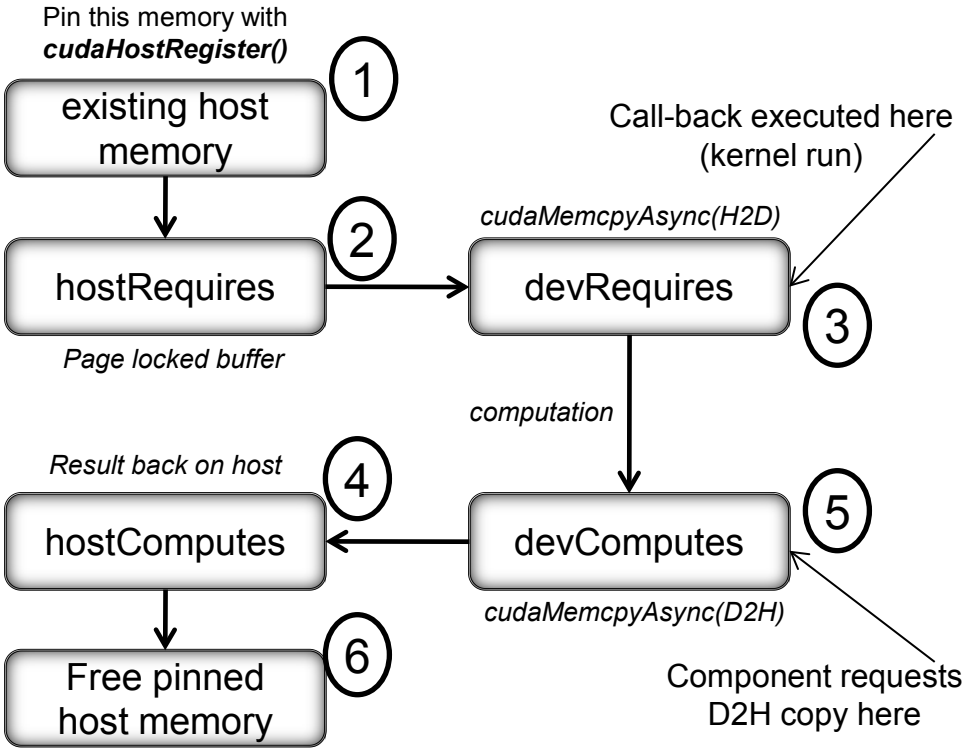
THESE MACHINES WILL SEEM "SMALL" IN 2025 and will the equivalent of large regional or lab machines but are ranked 2,7 and 4 in the world today

GPU Task Management



With Uintah's knowledge of the task-graph, task data can be automatically transferred asynchronously to the device before a GPU task executes

- All device memory allocations and asynchronous transfers handled automatically
- Can handle multiple devices on-node
- All device data is made available to component code via convenient interface



Memory Savings

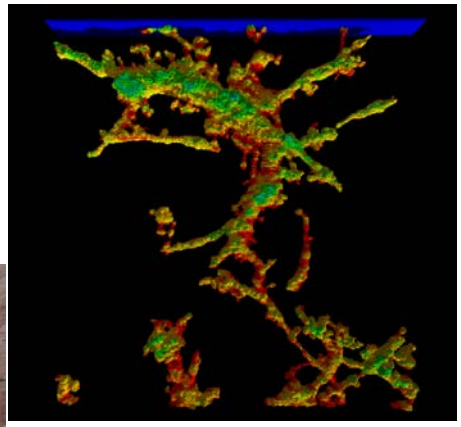
- Global Meta-data copies
  - 60 bytes or 7.5 doubles per patch
  - Each copy per core vs Each copy per node
- MPI library buffer overhead
- Results:

$$\text{Ratio} = \frac{\text{Thread MPI memory usage}}{\text{MPI memory usage}} \times 100\%$$

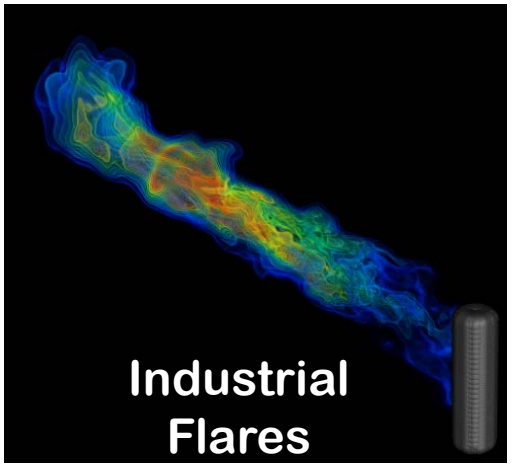
Cores	3072	6144	12288	24576	49152	98304
Percent	61%	47%	36%	27%	18%	11%

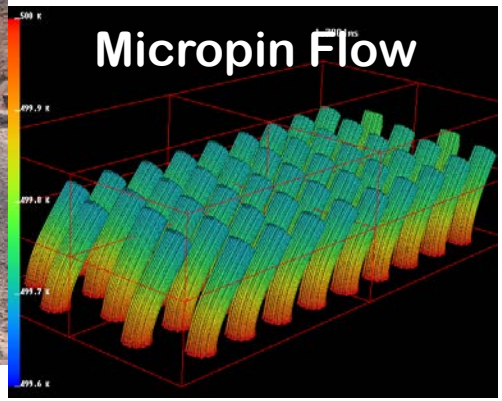
AMRICE: Simulation of the transport of two fluids with a prescribed initial velocity of Mach two: 435 million cells, strong scaling runs on Kraken

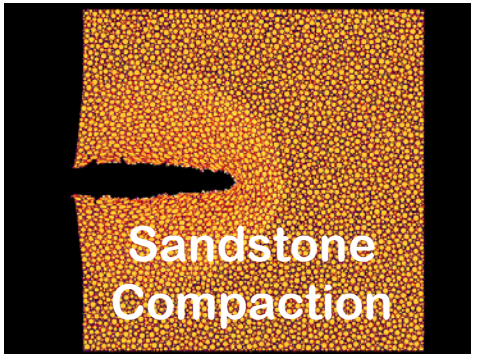
Uintah Applications



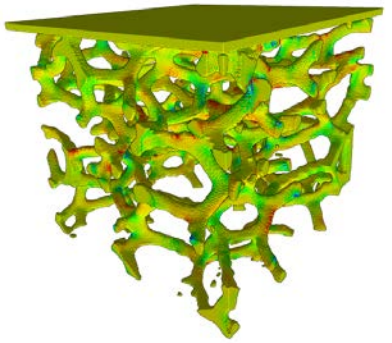
Angiogenesis

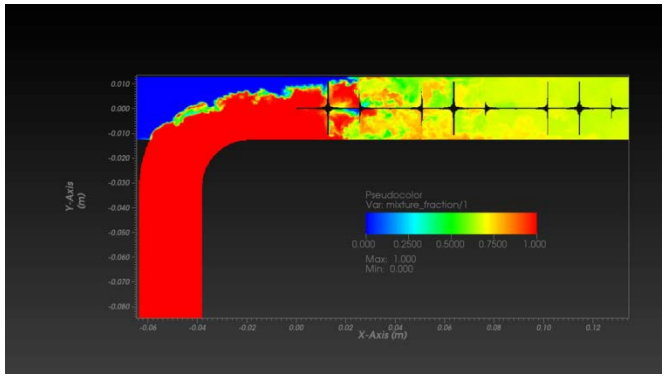






Foam
Compaction





Carbon capture and cleanup